An extended abstract of this paper appears in the proceedings of COCOON 2016. This is the full version.

Homomorphic Evaluation of Lattice-Based Symmetric Encryption Schemes

Pierre-Alain Fouque[1,3], Benjamin Hadjibeyli[2], and Paul Kirchner[3]

[1] Université de Rennes 1 and Institut Universitaire de France
[2] École normale supérieure de Lyon Benjamin.Hadjibeyli@ens-lyon.fr
[3] École normale supérieure {Pierre-Alain.Fouque,Paul.Kirchner}@ens.fr

Abstract. Optimizing performance of Fully Homomorphic Encryption (FHE) is nowadays an active trend of research in cryptography. One way of improvement is to use a hybrid construction with a classical symmetric encryption scheme to transfer encrypted data to the Cloud. This allows to reduce the bandwidth since the expansion factor of symmetric schemes (the ratio between the ciphertext and the plaintext length) is close to one, whereas for FHE schemes it is in the order of 1,000 to 1,000,000. However, such a construction requires the decryption circuit of the symmetric scheme to be easy to evaluate homomorphically. Several works have studied the cost of homomorphically evaluating classical block ciphers, and some recent works have suggested new homomorphic oriented constructions of block ciphers or stream ciphers. Since the multiplication gate of FHE schemes significantly increases the noise of the ciphertext, we cannot afford too many multiplication stages in the decryption circuit. Consequently, FHE-friendly symmetric encryption schemes have a decryption circuit with small multiplication depth.
We aim at minimizing the cost of the homomorphic evaluation of the decryption of symmetric encryption schemes. To do so, we focus on schemes based on learning problems: Learning With Errors (LWE), Learning Parity with Noise (LPN) and Learning With Rounding (LWR). We show that they have lower multiplicative depth than usual block ciphers, and hence allow more FHE operations before a heavy bootstrapping becomes necessary. Moreover, some of them come with a security proof. Finally, we implement our schemes in HElib. Experimental evidence shows that they achieve lower amortized and total running time than previous performance from the literature: our schemes are from 10 to 10,000 more efficient for the time per bit and the total running time is also reduced by a factor between 20 to 10,000. Of independent interest, the security of our LWR-based scheme is related to LWE and we provide an efficient security proof that allows to take smaller parameters.

1 Introduction

Fully Homomorphic Encryption (FHE) is nowadays one of the most active trend of research in cryptography. In a nutshell, a FHE scheme is an encryption scheme that allows evaluation of arbitrarily complex programs on encrypted data. This idea has been introduced by Rivest, Adleman and Dertouzos [32] in 1978, while the first plausible construction has been given by Gentry [18] in 2009. Since, numerous papers have focused on improving the efficiency of the constructions. Even if there still remains a huge gap to fill before FHE becomes practical, it arouses more and more interest and the scope of application is broad, going from genomics to finance [29]. Recently, the application of FHE techniques to genomics has been highlighted in the iDASH Privacy and Security Workshop with the Secure Genome Analysis Competition [4].

One way of improvement has been introduced in [29]. It focuses on minimizing the communication complexity of the scheme. The idea is to use a hybrid encryption scheme: some parts of the scheme are replaced by a symmetric encryption scheme. Instead of encrypting the

[4] http://www.humangenomeprivacy.org/2015/index.html

data under the FHE scheme, the client will only encrypt its symmetric key under the FHE scheme, and encrypt its data under the symmetric scheme. The cloud will then homomorphically evaluate the decryption of the symmetric scheme on the symmetrically encrypted data and the homomorphically encrypted symmetric key, to get a ciphertext corresponding to a homomorphic encryption of the data.

Clearly, such a construction has low communication complexity, since the only online data transfer is made under the symmetric scheme. However, the cloud might pay a huge cost at the homomorphic evaluation of the symmetric decryption. Thus, one can look for the most FHE-friendly symmetric encryption scheme to use in the hybrid construction.

Being FHE-friendly consists in optimizing several criteria. First, as the application we gave suggests, we want a scheme with a small expansion factor, so that the communication complexity stays low. Then, other criteria depend on the FHE construction we are building upon. All current FHE schemes are based on variants of Gentry's initial idea: ciphertext consists of encryption of data with noise, and homomorphic operations increase this noise. When the upper bound of noise is reached, one has to bootstrap, to reduce the noise to its initial level. Typically, functions are represented as arithmetic circuits and multiplications have a far higher cost than additions in terms of noise. Thus, we will want to minimize the multiplicative depth of the decryption circuit of our symmetric scheme. In addition, we will also take into account the total running time of our homomorphic evaluation step. This metric highly depends on the chosen FHE scheme, but multiplications often happen to be the main bottleneck again.

Our Contributions. In this paper, we focus on symmetric schemes having shallow decryption circuits. We study the problem of building secure symmetric encryption scheme with constant or small decryption circuit, namely with small multiplication depth. Contrary to the direction followed by many recent work, that tweak block ciphers or stream ciphers [3,11], our approach is related to provable security. Indeed, we notice that one can construct lattice-based schemes with very small decryption circuit and then, we evaluate the performances of our schemes using HElib to compare them with other symmetric ciphers. Finally, we try to use HElib features (full packing and parallelization) in order to achieve better performances. We describe two kinds of ciphers: the first family has its security related to the difficulty of solving the LPN problem in specific instances, while the second family has a *security proof* based on the LWE problem. The first construction is similar to symmetric cryptography since we do not have a clean security proof and consequently, we provide a more thorough security analysis. However, the security seems to be easier to understand than ad-hoc constructions usually used in symmetric cryptography, since the security problem on which the scheme is based can be formally stated. We present a very efficient construction specifically tailored to this problem to secure our construction from Arora-Ge type of attack on LPN. The performance of the schemes from this family can be 10 times more efficient than the most efficient previous cipher. For the second family, we have a rigorous security proof related to LWE, while the scheme is based on LWR. The performance of the second family can be very efficient, about 10,000 times faster, but the caveat is that the decrypted plaintext contains random bits in the least significant bits if we do not compute homomorphically the truncation using the costly $\mathsf{ExtractDigits}$ function. Therefore, if we want to remove the erroneous bits, the performances become equivalent to previous ciphers, while being more efficient than AES. In some cases, one can compute with such noise without having to remove it.

We notice that contrary to what is claimed in many works [29], it is not necessary to re-encrypt the symmetrically-encrypted ciphertext using the FHE scheme when the server receives the data. We show in appendix 3.2 that the evaluation of the homomorphic decryption procedure gives ciphertexts encrypted with FHE. This improves the performance of the scheme, since we homomorphically evaluate the function that maps the key K to the $\mathsf{Dec}(K, c)$, given the ciphertext c and some multiplications in Dec will be simplified once c is known.

Then, we describe our efficient FHE-friendly symmetric schemes. We chose to use schemes based on lattices, and more precisely on learning problems. Our results show that we can get circuits with very small multiplication depth for the decryption algorithms of these schemes. In addition, their security relies on hard problems or on hard instances of lattice problems in the worst cases, as opposed to usual block ciphers.

We first present a scheme whose security is based on the Learning Parity With Noise problem (LPN) introduced in [20]. We have to specify an error correcting code (ECC) for this scheme so that the decryption circuit is small. We choose to use a repetition code in order to simplify the decoding and reduce its circuit in term of multiplications [5]. However, removing decryption failures makes the scheme vulnerable to the Arora-Ge [6] attack and in order to avoid its most efficient variant [2] using Gröbner basis algorithms, we use a very efficient transformation, similar to random local function [5], which increases the algebraic degree of the polynomials system. We provide a detailed analysis of this attack. The function we propose is also very similar to [1] and we can show that our construction achieves better influence parameters, but it has higher complexity class since we need a logarithmic depth circuit.

Then, we introduce another scheme whose security is based on the Learning With Rounding problem (LWR) and a very similar version whose security relies directly on the Learning With Errors (LWE) problem. In order to encrypt many bits using small parameters, we provide a direct proof from LWE to the security of the scheme. We do not rely on any reduction from LWE to LWR since the first reduction given by Banerjee et al. [7] requires exponential parameters and the one by Alwen et al. [4] requires parameter linear in the number of samples. Here, our reduction is only logarithmic in the number of samples.

Furthermore, we extend both schemes to their ring versions. In this case, we optimized the number of multiplications using a FFT algorithm in order to compute the polynomial multiplications. Finally, we extend them to their module versions, which generalizes standard and ring versions.

Along with a theoretical analysis, we provide implementations of the homomorphic evaluation of our schemes in HElib, in order to make practical comparisons. While the homomorphic evaluation of AES went down to 11 milliseconds per bit [19] and LowMC, a block cipher designed to be FHE-friendly (and whose security has recently been called into question [13,14]), went down to 3 milliseconds per bit [3], which was the best so far, we go under a millisecond per bit (with the module version of our LPN scheme). In some scenario, our performance for the scheme based on LWR are drastically better if we allow FHE-encrypted plaintexts to contain noise in the least significant bits. Moreover, our schemes are a lot more flexible, in the sense that they need smaller FHE parameters, and while these performance were amortized over a computation taking several minutes, the evaluation of our schemes takes only from a second to a minute.

Related work. Numerous papers have presented homomorphic evaluations of block ciphers. It has started in [19], where AES has been chosen as a benchmark for measuring the performance of HElib. Then, performance has been improved in [28]. AES has then been used as benchmark for comparing FHE schemes in [12,15]. Similarly, Simon has been used to compare FHE schemes [26]. Recently, the problem has been taken the other way round, with works trying to find the most FHE-friendly block cipher. First, a lightweight block cipher like Prince has been suggested and evaluated [16]. Then, a new block cipher, LowMC, has been designed specifically for this kind of application [3], as well as for multi-party computations and zero-knowledge proofs. Finally, using stream ciphers has also been proposed [11].

Organization of the paper. In section 2 we recall definitions about symmetric encryption and Lattice problems in Cryptography. Then we introduce in section 4 our symmetric schemes based on learning problems: LPN, LWR and LWE. In App. 3.2, we explain how we

[5] More complex ECC exist with constant decoding such as [21] but they are only interesting from an asymptotic point of view.

use homomorphic operation more efficiently. The security analysis of the schemes are proved in the final version. In this version, we also describe the efficiency of our schemes.

2 Preliminaries

We recall basic definitions about symmetric encryption and lattice assumptions on which the security of our symmetric and homomorphic schemes is based.

2.1 Symmetric Encryption

We will say that a function of k (from positive integers to positive real numbers) is negligible if it approaches zero faster than any inverse polynomial, and noticeable if it is larger than some inverse polynomial (for infinitely many values of k).

Definition 1. *A symmetric encryption scheme is a tuple* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *of Probabilistic Polynomial-time (PPT) algorithms as follows:*

- $\mathsf{Gen}(1^\lambda)$*: given a security parameter* λ*, output a secret key* k*;*
- $\mathsf{Enc}(k, m)$*: given a key* k *and a message* m*, output a ciphertext* c*;*
- $\mathsf{Dec}(k, c)$*: given a key* k *and a ciphertext* c*, output a message* m'*;*

which satisfy the following correctness property: if $k := \mathsf{Gen}(1^\lambda)$*, then for all messages* m*,* $\Pr[\mathsf{Dec}(k, \mathsf{Enc}(k, m)) \neq m]$ *is negligible (in* λ*).*

For the sake of clarity, we will often write the key as a subscript and the scheme name as a superscript of our algorithms, like in Enc_k^S.

Definition 2. *A symmetric encryption scheme* S *is said to be semantically secure (or to have indistinguishable encryptions) if for all PPT* $\mathcal{A}$ *and for all messages* m*, it holds that*

$$\left| \Pr[\mathcal{A}(\mathsf{Enc}_k^S(m), 1^\lambda) = 1] - \Pr[\mathcal{A}(\mathsf{Enc}_k^S(0), 1^\lambda) = 1] \right|$$

is negligible (in λ*).*

Indeed, semantic security is implied by the following property, which will be satisfied by our schemes.

Definition 3. *A symmetric encryption scheme* S *is said to have pseudo-random ciphertexts (or to have ciphertexts indistinguishable from random) if no PPT* $\mathcal{A}$ *can distinguish between ciphertexts from the scheme and the uniform distribution, i.e. for all PPT* $\mathcal{A}$ *and for all messages* m*, it holds that*

$$\left| \Pr[\mathcal{A}(\mathsf{Enc}_k^S(m), 1^\lambda) = 1] - \Pr[\mathcal{A}(r, 1^\lambda) = 1] \right|$$

is negligible (in λ*), where* r *is drawn randomly over the ciphertext space.*

Most symmetric encryption schemes are based on a construction called block cipher. A block cipher is a keyed permutation working on fixed-length block. Formally, a block cipher E takes as input a key K of length k and a string m of length n and outputs a string c of length n. It is required that each function $E_K : m \to E(K, m)$ is invertible. To construct an encryption scheme from these building blocks, one has to define what is called a mode of operation.

2.2 Learning Problems

Given a finite set S and a probability distribution D on S, $s \leftarrow D$ denotes the drawing of an element of S according to D and $s \leftarrow S$ denotes the random drawing of an element of S endowed with the uniform probability distribution.

Learning With Errors. The Gaussian distribution with standard deviation σ is defined on $\mathbb{R}$ by the density function $\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{1}{2}(\frac{x}{\sigma})^2)$.

The Learning With Errors problem (LWE) has been introduced in [31]. For $s \in \mathbb{Z}_q^k$, the LWE distribution $D_{s,\chi}^{LWE}$ is defined over $\mathbb{Z}_q^k \times \mathbb{Z}_q$ and consists in samples $(a, \langle a, s\rangle + e)$ where $a \leftarrow \mathbb{Z}_q^k$ and $e \leftarrow \chi$ for some distribution χ over $\mathbb{Z}_q$. Typically, χ is taken to be some integral Gaussian distribution when assuming that LWE is hard. As in most works [31], we will consider here rounded Gaussian distributions: it basically consists in sampling a Gaussian distribution, reducing the result modulo 1, multiplying it by q and rounding it to the nearest integer. LWE consists, for s chosen according to some distribution over $\mathbb{Z}_q^k$ (typically, the uniform distribution), in distinguishing between any desired number of samples from $D_{s,\chi}^{LWE}$ and the same number of samples drawn from the uniform distribution over $\mathbb{Z}_q^k \times \mathbb{Z}_q$. For rounded Gaussian distributions, LWE is usually considered to be hard when the standard deviation σ verifies $\sigma > \sqrt{k}$ [31].

LWE can be extended into a ring version RLWE [27]. Let $R = \mathbb{Z}[X]/(P(X))$ for a monic irreducible polynomial P of degree k, and let $R_q = R/qR$. Generally, P is chosen to be some power-of-two cyclotomic polynomial, which are of the form $X^{2^z}+1$. For an element $s \in R_q$, we define the RLWE distribution $D_{s,\chi}^{RLWE}$ over $R_q \times R_q$ by samples $(a, a.s+e)$ where $a \leftarrow R_q$ and $e \leftarrow \chi^k$ where χ^k consists in k independent samples from χ and e is interpreted as an element of R_q. The Ring-LWE problem (RLWE) similarly consists, for s drawn according to some distribution over R_q, in distinguishing $D_{s,\chi}^{RLWE}$ from the uniform distribution over $R_q \times R_q$.

We will also use the Module-LWE problem (MLWE). It has been introduced in [10] under the name of GLWE, for General LWE. However, we will call it MLWE as in [25], because it indeed corresponds to introducing a module structure over LWE. For an element $s \in R_q^k$, where the underlying ring polynomial has degree d, we define the MLWE distribution $D_{s,\chi}^{MLWE}$ over $R_q^k \times R_q$ by samples $(a, \langle a.s\rangle + e)$ where $a \leftarrow R_q^k$ and $e \leftarrow \chi^d$ is interpreted as an element of R_q. MLWE generalizes LWE and RLWE: LWE corresponds to the case where $d = 1$ and RLWE corresponds to the case where $k = 1$.

By a standard hybrid argument, LWE can be extended to several secrets. It can be shown that the problem which consists in distinguishing samples $(a, \langle a, s_1\rangle + e_1, \ldots, \langle a, s_n\rangle + e_n) \in \mathbb{Z}_q^k \times \mathbb{Z}_q^n$ from the uniform distribution over $\mathbb{Z}_q^k \times \mathbb{Z}_q^n$, where each $s_j \in \mathbb{Z}_q^k$ is chosen independently for any $n = poly(k)$, is at least as hard as LWE for a single secret s. An analogous statement can be shown for RLWE and MLWE.

Finally, the LWE [31], RLWE [27] and MLWE [25] hardness assumptions have been reduced to standard lattice assumptions. The security of MLWE seems to be intermediate between the security of LWE based on hardness results in arbitrary lattices and the security of RLWE in ideal lattices.

Learning Parity With Noise. We denote by $\mathcal{B}_\eta$ the Bernoulli distribution of parameter $\eta \in [0, 1]$, i.e. a bit $b \leftarrow \mathcal{B}_\eta$ is chosen such that $\Pr[b = 1] = \eta$ and $\Pr[b = 0] = 1 - \eta$. The Learning Parity with Noise problem (LPN) basically consists in LWE for $q = 2$. The distribution χ chosen over $\mathbb{Z}_2$ will correspond to a Bernoulli distribution. We extend LPN to RLPN and MPLN. The only difference is that the underlying polynomial will not be cyclotomic anymore, but some irreducible polynomial modulo 2. Similarly, these problems are also extended to a polynomial number of secrets. The main difference between LWE and LPN is that the security of LPN remains heuristic because no reduction has been made so far to lattice problems.

Learning With Rounding. The Learning With Rounding problem (LWR) has been introduced in [7] as a derandomization of LWE. The idea is to replace the addition of a random noise by a rounding function.

Let k be the security parameter and moduli $q \geq p \geq 2$ be integers. We define the function $\lfloor . \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ by $\lfloor x \rceil_p = \lfloor (p/q).\bar{x} \rceil$, where $\bar{x}$ is an integer congruent to x mod q.

We extend $\lfloor . \rceil_p$ component-wise to vectors and matrices over $\mathbb{Z}_q$. Let R denote the cyclotomic polynomial ring $R = \mathbb{Z}[z]/(z^k + 1)$ for k a power of two. For any modulus q, we define the quotient ring $R_q = R/qR$ and extend $\lfloor . \rceil_p$ coefficient-wise to it. Note that we can use any common rounding method, like the floor or ceiling functions. In our implementations, we will indeed use the floor functions, because it is equivalent to dropping the least-significant digits in base 2 when q and p are both powers of 2.

For a vector $s \in \mathbb{Z}_q^k$, the LWR distribution D_s^{LWR} is defined over $\mathbb{Z}_q^k \times \mathbb{Z}_p$ by elements $(a, \lfloor \langle a, s \rangle \rceil_p)$ with $a \leftarrow \mathbb{Z}_q^k$. For a vector $s \in R_q$, the ring-LWR (RLWR) distribution D_s^{RLWR} is defined over $R_q \times R_p$ by elements $(a, \lfloor a.s \rceil_p)$ with $a \leftarrow R_q$. And for a vector $s \in R_q^k$, the module-LWR (MLWR) distribution D_s^{MLWR} is defined over $R_q^k \times R_p$ by elements $(a, \lfloor \langle a.s \rangle \rceil_p)$ with $a \leftarrow R_q^k$.

For a given distribution D over $s \in \mathbb{Z}_q^k$, LWR consists in distinguishing between any desired number of independent samples from D_s^{LWR} and the same number of samples drawn uniformly and independently from $\mathbb{Z}_q^k \times \mathbb{Z}_p$. RLWR and MLWR are defined analogously. All these problems can be extended to several secrets, as stated for LWE.

The security of LWR has been reduced to the one of LWE when q/p is exponential in k by [7], and when q/p is polynomial in k and linear in the number of samples by [4].

3 Fully-Homomorphic Encryption (FHE)

While classical encryption aims at preserving the privacy of information, homomorphic encryption aims, in addition, at making some computation on the encrypted data.

3.1 FHE Definitions

Formally, we have a message space M with a set of functions f we would like to compute on messages, and we want an algorithm which efficiently computes functions f' on the ciphertext space C such that $\mathsf{Dec}(f'(\{c_i\}_i)) = f(\{\mathsf{Dec}(c_i)\}_i)$. Thus, we want the decryption function to be a homomorphism from C to M for these functions f. This notion, originally called a privacy homomorphism, was introduced in[32]. Here is a formal definition of a homomorphic scheme, sometimes also referred to as "somewhat homomorphism".

Definition 4. *Let $\mathcal{F}$ be a set of functions. A $\mathcal{F}$-homomorphic encryption (HE) scheme is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Eval})$ *as follows:*

- $\mathsf{Gen}(1^\lambda)$*: given a security parameter λ, output a public key pk, a secret key sk and an evaluation key ek;*
- $\mathsf{Enc}(pk, m)$*: given the public key pk and a message m, output a ciphertext c;*
- $\mathsf{Dec}(sk, c)$*: given the secret key sk and a ciphertext c, output a message m';*
- $\mathsf{Eval}(ek, f, \Psi = (c_1, \ldots, c_l))$*: given the evaluation key, a function f and a tuple Ψ of l ciphertexts, where l is the arity of f, output a ciphertext c';*

and which satisfies the correctness property: for all functions $f \in \mathcal{F}$ and messages $\{m_i\}_{i \leq l}$, where l is the arity of f, if $(pk, sk, ek) := \mathsf{Gen}(1^\lambda)$ and $c_i := \mathsf{Enc}(pk, m_i)$ for all i, then $\Pr[\mathsf{Dec}(sk, \mathsf{Eval}(ek, f, (c_1, \ldots, c_l))) \neq f(m_1, \ldots, m_l)]$ *is negligible (in λ).*

Definition 5. *A HE scheme H is said to be semantically secure if for all PPT $\mathcal{A}$ and for all messages m, it holds that*

$$\left| \Pr[\mathcal{A}(\mathsf{Enc}_k^H(m), 1^\lambda) = 1] - \Pr[\mathcal{A}(\mathsf{Enc}_k^H(0), 1^\lambda) = 1] \right|$$

is negligible (in λ).

Full homomorphism is a generalization of this concept: in a nutshell, a fully homomorphic encryption scheme is an encryption scheme that allows evaluation of arbitrarily complex functions on encrypted messages. The first construction of such a scheme appeared in 2009 [18] and was a major breakthrough.

However, being simply homomorphic for all functions is not enough, since it permits the trivial solution, where c' consists of (f, Ψ) and Dec decrypts the ciphertexts and then applies f. A way of excluding the trivial solution is to require that the distribution of the ciphertexts output by Eval is statistically close to the one of Enc: this notion is called circuit privacy. However, most of the literature aims at achieving a weaker property.

Definition 6. *A HE scheme is said to be compact if the length of the output of the algorithm* Eval *is polynomial in the security parameter* λ*.*

Definition 7. *A Fully Homomorphic Encryption (FHE) scheme is a scheme which is compact and* $\mathcal{F}$*-homomorphic for any class* $\mathcal{F}$ *of functions.*

In this work, f will be represented by an arithmetic circuit, as in most of the literature. Even if we will most of the time use circuits over $GF(2)$, we will sometimes use circuits over over $\mathbb{Z}_q$, for example when dealing with LWR. The following definition is a slight relaxation of FHE.

Definition 8. *A Leveled Homomorphic Encryption (LHE) scheme is a compact* C^L*-homomorphic scheme* $\mathcal{HE}$ *where* $\mathsf{Gen}^{\mathcal{HE}}$ *gets an additional input* 1^L *and where* C^L *is the class of all depth-*L *arithmetic circuits.*

Basically, a homomorphic scheme is leveled if for any depth L, we can fix the parameters so that any circuit of depth at most L can be evaluated. But unlike a FHE scheme, the complexity of the algorithms in a LHE might depend on this maximal depth. The BGV scheme we will present in section 5 is indeed a LHE. It becomes a FHE when considered along with its bootstrapping functionality.

3.2 Homomorphic evaluation of symmetric encryption schemes

We now give a more precise description of the scenario where a symmetric encryption scheme is used to improve FHE performance, as described in [29], and on which we will to rely to analyse the performance of our schemes.

Optimizing communication with the cloud. Consider the setting where a client uploads its data encrypted under a FHE scheme on a cloud service and wants the cloud to compute on this data and return encrypted outputs. Typically, FHE schemes come with an expansion factor of the order of 1,000 to 1,000,000. To mitigate this problem, the client will send its data encrypted under some semantically secure symmetric encryption scheme (which, by itself, is not homomorphic at all) along with the homomorphic encryption of its symmetric key. Then, the steps of symmetric decryption can all be carried out on homomorphically encrypted entries. Thus, the cloud can obtain the data encrypted under the FHE scheme by homomorphically evaluating the decryption circuit.

Here is a formal description of the protocol. Let $H = (\mathsf{Gen}^H, \mathsf{Enc}^H, \mathsf{Dec}^H, \mathsf{Eval}^H)$ be a FHE scheme and let $S = (\mathsf{Gen}^S, \mathsf{Enc}^S, \mathsf{Dec}^S)$ be a symmetric encryption scheme. Let λ be the security parameter and m be the data the client wants to send to the cloud. Let $(pk, sk, ek) := \mathsf{Gen}^H(1^\lambda)$ and $k := \mathsf{Gen}^S(1^\lambda)$.

- The client first sends the messages $c_1 := \mathsf{Enc}^H_{pk}(k)$ and $c_2 := \mathsf{Enc}^S_k(m)$ to the cloud.
- Given a couple of ciphertexts (c_1, c_2) received from the client, the cloud will compute (either at the reception or just before further computing) $x = \mathsf{Enc}^H_{pk}(c_2)$ and then $c = \mathsf{Eval}_{ek}(\mathsf{Dec}^S, c_1, x)$: this is the step where we need an efficient homomorphic evaluation of the decryption circuit of our symmetric scheme.

– Now, the cloud possesses a FHE-encrypted ciphertext c, which means that $\mathsf{Dec}^H_{sk}(c) = m$. Furthermore, it can now homomorphically evaluate any function f: for all f, $\mathsf{Dec}^H_{sk}(\mathsf{Eval}^H_{ek}(f, c)) = f(c)$.

Indeed, if the evaluation algorithm allows constant arguments, i.e. arguments which are not homomorphically encrypted, this scenario can be optimized further, simply by noticing that c_2 does not have to be homomorphically encrypted. Thus, when receiving c_1 and c_2, the cloud will just directly compute $c = \mathsf{Eval}_{ek}(\mathsf{Dec}^S, c_2, c_1)$. It still has to homomorphically evaluate the decryption circuit of the symmetric scheme, but it saves a homomorphic encryption, and operations with constants might be faster. This can also be seen as evaluating the function $K \mapsto Dec^S(K, m)$, which depends on m.

The total communication complexity of this protocol is the sum of lengths of the public key, homomorphically encrypted symmetric key and symmetrically encrypted data. However, the public key and symmetric key are sent only once, are independent of the data, and can be used for many messages. Therefore, they do not have to be taken into account for the online complexity of the protocol, and only the expansion factor of the symmetric schemes is relevant.

Others usages. Even if we focus on homomorphic encryption, note that the schemes we present could be interesting for some other applications which have the same kind of constraints, like multi-party computation [3] or encryption schemes secure against side-channel attacks. Indeed, in this latter area, researchers try to reduce the nomber of non-linear operations since these operations are costly when we use masking. In our schemes, the number of non-linear operations per bits is a small constant.

4 FHE-friendly Symmetric Encryption based on Learning

All the symmetric encryption schemes we will use are based on lattices, and, more precisely, on learning problems. Some of them will rely on the LPN problem, while the others will rely on the LWR or on the LWE problem.

4.1 An encryption scheme based on MLPN-C

Our first encryption scheme is a generalization of the scheme introduced in [20], under the name of $\mathtt{LPN}-\mathtt{C}$ by Gilbert *et al.* Let $[n, m, d]$ be a linear binary (error-correcting) code C, i.e. a linear subspace of $\mathbb{F}_2^n$ with dimension m such that d is the minimum ℓ^1 distance between two elements of the code. We associate it with an encoding function $E : \mathbb{F}_2^m \to C$ and a decoding function $D : C \to \mathbb{F}_2^m$. $\mathtt{LPN}-\mathtt{C}$ is a symmetric encryption scheme whose security can be reduced to the hardness of LPN. Let E and D be respectively the encoding and decoding functions of a $[n, m, d]$ linear binary code.

Definition 9 ($\mathtt{LPN}-\mathtt{C}$). *Let k and n be polynomials in λ. The symmetric encryption scheme $\mathtt{LPN}-\mathtt{C}$ is defined in the following way:*

– $\mathsf{Gen}(1^\lambda)$*: output* $S \leftarrow \mathbb{Z}_2^{k\times n}$*;*
– $\mathsf{Enc}_S(x)$*: output* $(a, E(x) \oplus a.S \oplus e)$*, where* $a \leftarrow \mathbb{Z}_2^k$ *and* $e \leftarrow \mathcal{B}_\eta^n$*;*
– $\mathsf{Dec}_S(a, y)$*: output* $D(y \oplus a.S)$*.*

For a message of m bits, this scheme produces a ciphertext of $n + k$ bits. Indeed, the expansion factor can tend to the one of the linear code we are using, which is n/m, since n is any polynomial in k. Furthermore, one can consider that a does not have to be sent, and can be replaced, for example, by the seed used in order to generate it.

The decryption requires k multiplications for the scalar product, in addition to the multiplications required by the decoding, but the multiplicative depth of the scalar product

is only 1, to which we have to add the depth of the decoding circuit. Furthermore, as explained in subsection 3.2, the vector a can be considered as a constant, and thus, the level of multiplication corresponding to the scalar product prior to the decoding counts for less than one.

We introduce the ring version of $\mathtt{LPN}-\mathtt{C}$. We extend functions E and D over R_2^m by applying them on vectors corresponding to coefficients of same degree.

Definition 10 ($\mathtt{RLPN}-\mathtt{C}$)**.** *Let k and n be polynomials in λ. Let consider R chosen as in RLPN, with underlying polynomial P of degree k. The symmetric encryption scheme $\mathtt{RLPN}-\mathtt{C}$ is defined in the following way:*

- $\mathsf{Gen}(1^\lambda)$*: output* $S \leftarrow R_2^n$*;*
- $\mathsf{Enc}_S(x)$*: output* $(a, E(x) \oplus a.S \oplus e)$*, where* $a \leftarrow R_2$ *and* $e \leftarrow \mathcal{B}_\eta^{k\times n}$ *is interpreted as an element of* R_2^n*;*
- $\mathsf{Dec}_S(a, y)$*: output* $D(y \oplus a.S)$*.*

For a message of $m \times k$ bits, the scheme produces a ciphertext of $k + k \times n$ bits. Thus, the expansion factor tends again to the one of the underlying linear code. The decryption depth is also similar to the one of $\mathtt{LPN}-\mathtt{C}$ (a polynomial multiplication has multiplicative depth 1). The main difference in the ring version is the number of bits encrypted for a key of fixed size: while we were encrypting m bits at once in the standard version, we can now encrypt $m \times k$ bits at once.

Similarly, we extend the scheme to MLPN.

We describe the more general version of our scheme. Similarly, we can define $\mathtt{LPN}-\mathtt{C}$ and $\mathtt{RLPN}-\mathtt{C}$ based on LPN and RLPN problems.

Definition 11 ($\mathtt{MLPN}-\mathtt{C}$)**.** *Let d, k and n be polynomials in λ. Let consider $\mathbb{F}_{2^d}$ a finite field defined by an irreducible polynomial P of degree d. The symmetric encryption scheme $\mathtt{MLPN}-\mathtt{C}$ is defined in the following way:*

- $\mathsf{Gen}(1^\lambda)$*: output* $S \leftarrow \mathbb{F}_{2^d}^{k\times n}$*;*
- $\mathsf{Enc}_S(x)$*: output* $(a, E(x) \oplus a.S \oplus e)$*, where* $a \leftarrow \mathbb{F}_{2^d}^k$ *and* $e \leftarrow \mathcal{B}_\eta^{d\times n}$ *is interpreted as an element of* $\mathbb{F}_{2^d}^n$*;*
- $\mathsf{Dec}_S(a, y)$*: output* $D(y \oplus a.S)$*.*

For a message of m bits, this scheme produces a ciphertext of $n + k$ bits. Indeed, the expansion factor can tend to the one of the linear code we are using, which is n/m, since n is any polynomial in k. Furthermore, one can consider that a does not have to be sent, and can be replaced, for example, by the seed used in order to generate it.

We choose the 3-repetition code to have a small multiplication depth circuit of degree 2. We define the encoding scheme for $a \in \mathbb{F}_{2^d}$ as $(a^{2^{d-1}}, a^{2^{d-1}}, a^{2^{d-1}})$. In order to decode a code word $(a, b, c) \in (\mathbb{F}_{2^d})^3$, we compute $ab + bc + ac$. (The normal encoding with would be (a, a, a) and the decoding $(ab + ac + bc)^{2^{d-1}}$, but we prefer to incorporate the power 2^{d-1} in the encoding in order to make the homomorphic part more efficient.)

Proposition 1 ([20]). $\mathtt{LPN}-\mathtt{C}$ *(resp.* $\mathtt{RLPN}-\mathtt{C}$*,* $\mathtt{MLPN}-\mathtt{C}$*) is semantically secure as soon as the corresponding LPN (resp. RLPN, MLPN) problem is hard.*

Proof. While the LPN case is proven in [20], the RLPN and MLPN cases follow similarly from the RLPN and MLPN assumptions. Furthermore, one can notice that as soon that the LPN (or RLPN or MLPN) problem for several secrets is considered to be hard, as we said via a hybrid argument, the ciphertext of our scheme consists in some value added to some sample from this distribution, which is indistinguishable from uniform by assumption, and thus, the ciphertext is indistinguishable from uniform. □

Variant 1. As it stands, if we do not bound the number of errors sent along with a message, this scheme will produce decryption failures. They will happen when the Hamming weight of the noise vector e is greater than the correction capacity of the error-correcting code. We study the probability of decryption failures and we can choose the noise parameter η so that this probability is very low. However, in this case, more efficient attacks than BKW algorithm $O(2^{k/\log(k/-\log(1-2\eta))})$ can be used to recover the secret in time $O(k^3/(1-\eta)^k)$. To thwart attacks, we will increase their complexity using *delinearization steps* described later.

Variant 2. A way to prevent decryption failures is to test the Hamming weight of the noise vector before using it. If it is too large, the sender draws a new noise vector. This implies a longer encryption running time, but it is only on the client side and not in the homomorphic part (and in practice, we choose our parameters so that this time remains reasonable). However, such a modification gives a structure to the noise and we cannot reduce the security of the scheme to the one of LPN anymore. It makes the scheme vulnerable to the Arora-Ge attack, and we will study the security in the following section.

An important point is the choice of the error-correcting code used in the scheme. In our context, we would like a code with shallow decoding circuit, and indeed, codes with shallow decoding circuits are quite rare. For example, linear codes, which have really simple encoding circuits, have complicated decoding circuits. We would like to use a 3-repetition code, which has decoding depth 1. We will keep using such a code in practice as it leads to very efficient homomorphic performance, but the particular structure given to the noise requires a careful analysis of the security, that we will do in the following section. Consequently, in order to also thwart this attack, the delinearization steps can be useful.

Delinearization steps. In order to counter the Arora-Ge attack, we choose to add some noise on our values after computing the scalar product. In practice, the ciphertext we send consists of $(a, E(x) \oplus F(a.S) \oplus e)$ where F is some function involving enough layers of multiplication so that the Arora-Ge attack does not work. Of course, this step increases the parameters we have to choose for homomorphically evaluating our scheme, however, a few steps (3 in order to have a sufficient security parameter) are needed in order to prevent the attacks. We admit that such techniques are far away from provable security and come from symmetric cryptography since F is a kind of cheap non-linear operation. However, contrary to the symmetric setting, here the adversary cannot control the inputs to this function and many well-known chosen plaintext attacks are then prohibited and only known plaintext attacks need to be studied. It is easy to see how to adapt the decryption process.

The function F we choose works as follows: on a vector V, it consecutively applies several transformations T_i, for $i \leq d$, such that $[T_i(V)]_j = V_j + V_{x_{ij}} \cdot V_{y_{ij}}$, where the set of indices x_{ij} and y_{ij} is chosen so that monomials do not cancel. The degree of F in the inputs is 2^d. We estimated the number of applications of such transformations needed in order to counteract the most efficient variant of the Arora-Ge attack and for $n = 512$, two steps seem reasonable.

Even though, our scheme has a security proof, the parameters we choose do not allow us to use the reduction. Indeed, we pick either a structured noise (and Arora-Ge algorithms must be taken into account) or a very small noise to reduce the decryption failure. Consequently, the delinearization steps we add increase the complexity of these attacks and a thorough security analysis is presented in the next subsection.

4.2 Cryptanalysis of MLPN − C

In this section, we study the security of our schemes against different attacks. We have to do this analysis for two main reasons. First, even if our schemes are lattice-based and their security can be reduced to some hard problems, we have to look at the most efficient attacks

which are known in order to determine concrete parameters and [17] is needed for LWR. Second, we saw that in our scheme based on LPN, handling decryption failures precludes any reduction and creates vulnerabilities, so we have to look at the different possible attacks.

In what follows, we assume that the code we choose for our schemes based on LPN is a 3-repetition code. This choice is justified by the performance of our homomorphic evaluation.

Arora-Ge attack against Variant 2. The Arora-Ge attack consists in finding polynomials in the messages which vanishes on the secret key and then to solve the system of equations we get to determine the key. If we want to avoid decryption failures, we will require to have error vectors with at most t positive terms, where t is the correction capacity of the code, and thus any products of $t+1$ variables in the $y_i - \langle a, S_i \rangle - [E(x)]_i$ will vanish. From [6], we get that the complexity of such an attack is $poly(n^{t+1})$. The performance of the attack can be improved using Gröbner basis [2]. For a system of m equations of degree d over n variables, one defines what is called a "degree of regularity" D, and the complexity becomes $O(\binom{k+D}{k}^{\omega})$, where ω is the constant of the linear algebra.

Given the complexity of the attack, a simple countermeasure would be to pick a code with higher correcting capacity. However, finding a code with higher capacity and which is easy to evaluate homomorphically proved difficult. In order to prevent this attack, we prefer to use delinearization steps at the end.

Arora-Ge Attack against Variant 1. If we consider that controlling the error during the encryption leads to security holes, one could instead try to diminish the ratio of decryption failures. As we showed, decryption failures when the Hamming weight of the noise vector is greater than the correction capacity of the code. In other words, we can either decrease the ratio of error or increase the minimal distance of the code. We reckoned that modifying the code would worsen our homomorphic performance. On the other hand, we thought that the delinearization steps, introduced to counter the Arora-Ge attack, increased the security against BKW and gave some room for improvement on the error side. BKW works over equations which are linear in the secret variables, but the delinearization steps increase the degree of each term. Thus, in order to work on linear terms, one will have first to introduce new variables for each monomial, leading to an increase of the number of variables and a higher global complexity. Thus, instead of working on vectors of length k, BKW would work on vectors of size roughly k^{2^d}, where d is the number of delinearization step.

Now we will study the following attack, which is similar to Arora-Ge:

- Collect m equations $(y_i - F(\langle a, S_i \rangle) - [E(x)]_i) = 0$ and try to solve the corresponding system;
- In case of failure, restart.

The first step fails if and only if one of the errors used to generate the values y_i is not null. Let η be the probability that an error is not null. Then, the probability that the first step works is $(1-\eta)^m (\approx e^{-\eta m})$ and the number of repetitions is $(1-\eta)^{-m} (\approx e^{\eta m})$. Now, in order to solve the system, the most efficient method is to use Gröbner basis. From [8], we have that the complexity of such a computation is $O(\binom{k+D}{k}^{\omega})$, where D is called degree of regularity of the system and depends on m, k and d the degree of the equations we have in our system. So, for fixed values of m, k and d, we can determine what probability η we need in our scheme for x bits of security: it satisfies $e^{\eta m}\binom{k+D}{D}^{\omega} = 2^x$.

For $d = 2^2$ and $k = 80$ with a *large* field, the degree of regularity 5 is reached for $m \approx 390000$, and thus we get $\eta = \log(2^{80}/\binom{85}{5}^2)/(390 \cdot 10^3) \approx 6 \cdot 10^{-5}$, which gives a decryption failure every 40 MB. For $d = 2^2$ and $k = 512$, the degree of regularity 5 is reached for $m \approx 560 \cdot 10^6$, and thus we get $\eta = \log(2^{80}/\binom{517}{5}^2)/(560 \cdot 10^6) \approx 4 \cdot 10^{-9}$, which gives very few decryption failures (less than one every 10^{16} bits since when using a 3-repetition code, the error probability is η^2).

Linear Cryptanalysis. We need to assess the resistance of our scheme against linear cryptanalysis since it is the most efficient known-plaintext attack. The delinearization steps make the bias constant for this attack, however since the size of the LPN problem is very large, the problem the adversary tries to solve is a LPN problem and we can estimate its complexity. One delinearization step is the following non-linear operation $(a, b, c) \mapsto a + bc$ where a, b, c are bits. We can assume that each bit a, b and c is uniform under the LPN assumption and then, the best linear approximation is a with bias $1/4$. For two such steps, if the inputs are independent, the bias would be around $(1/4)^{2^1} = 2^{-4}$. Using the most efficient algorithm to solve this LPN problem [22], we can estimate that the complexity is at least 2^{80} if $n = 512$ (it would be 2^{-80} even if the bias was $1/8$).

On a field $\mathbb{F}_q$, the bias is $1/q - 1/q^2$ on 0 so for a large field, the noise is quickly negligible.

4.3 An encryption scheme based on LWR

We introduce a symmetric encryption scheme whose security can be reduced to the hardness of the LWE problem. We present the more general form $\mathtt{MLWR-SYM}$, but we can similarly define $\mathtt{LWR-SYM}$ and $\mathtt{RLWR-SYM}$.

Definition 12 ($\mathtt{LWR-SYM}$)**.** *Let k and n be polynomials in λ. The symmetric encryption scheme $\mathtt{LWR-SYM}$ is defined in the following way:*

- $\mathsf{Gen}(1^\lambda)$*: output* $S \leftarrow \mathbb{Z}_q^{k \times n}$*;*
- $\mathsf{Enc}_S(x)$*: output* $(a, x + \lfloor a.S \rceil_p)$*, where* $a \leftarrow \mathbb{Z}_q^k$*;*
- $\mathsf{Dec}_S(a, y)$*: output* $y - \lfloor a.S \rceil_p$*.*

For a message of size n over $\mathbb{Z}_p$, this scheme produces a ciphertext consisting of a random vector of length k over $\mathbb{Z}_q$ and a vector of length n of $\mathbb{Z}_p$. Thus, the expansion factor is $1 + \frac{\log q}{\log p}\frac{k}{n}$. Now, for the same reasons as for $\mathtt{LPN-C}$, this expansion factor can basically be considered as 1. The decryption circuit has depth one plus the depth of the rounding function. When using the floor function and if q and p are power of two, then the rounding consists in dropping the least significant bits of the result.

As for LPN, we introduce its ring and module versions.

Definition 13 ($\mathtt{RLWR-SYM}$)**.** *Let k and n be polynomials in λ. Let consider R as in RLWR, with underlying polynomial P of degree k. The symmetric encryption scheme $\mathtt{RLWR-SYM}$ is defined in the following way:*

- $\mathsf{Gen}(1^\lambda)$*: output* $S \leftarrow R_q$*;*
- $\mathsf{Enc}_S(x)$*: output* $(a, x + \lfloor a.S \rceil_p)$*, where* $a \leftarrow R_q$*;*
- $\mathsf{Dec}_S(a, y)$*: output* $(y - \lfloor a.S \rceil_p)$*.*

Definition 14 ($\mathtt{MLWR-SYM}$)**.** *Let d, k and n be polynomials in λ. Let consider R, with underlying polynomial P of degree d. The symmetric encryption scheme $\mathtt{MLWR-SYM}$ is defined in the following way:*

- $\mathsf{Gen}(1^\lambda)$*: output* $S \leftarrow R_q^k$*;*
- $\mathsf{Enc}_S(x)$*: output* $(a, x + \lfloor a.S \rceil_p)$*, where* $a \leftarrow R_q^k$*;*
- $\mathsf{Dec}_S(a, y)$*: output* $(y - \lfloor a.S \rceil_p)$*.*

For a message of size n over $\mathbb{Z}_p$, this scheme produces a ciphertext consisting of a random vector of length k over $\mathbb{Z}_q$ and a vector of length n of $\mathbb{Z}_p$. Thus, the expansion factor is $1 + \frac{\log q}{\log p}\frac{k}{n}$. Now, for the same reasons as for $\mathtt{LPN-C}$, this expansion factor can basically be considered as 1. The decryption circuit has depth one plus the depth of the rounding function. When using the floor function and if q and p are power of two, then the rounding consists in dropping the least significant bits of the result. The most efficient FHE-friendly encryption scheme works in only adding the plaintext on the $\log p$ most significant bits of $\lfloor a.S \rceil_p$ to avoid the costly ExtractDigits homomorphic function and the returned plaintext contains noise.

Proposition 2. $\mathtt{LWR}-\mathtt{SYM}$ *(resp.* $\mathtt{RLWR}-\mathtt{SYM}$, $\mathtt{MLWR}-\mathtt{SYM}$*) is semantically secure as soon as the corresponding LWR (resp. RLWR, MLWR) problem is hard.*

Proof. By the LWR assumption, $(a, \lfloor a.S \rceil_p)$ is produced according to a distribution which is indistinguishable from the uniform distribution. Thus, adding the plaintext to the second term still produces some couple which is indistinguishable from uniform. The argument holds for RLWR and MLWR. □

4.4 An encryption scheme based on LWE

We can adapt $\mathtt{LWR}-\mathtt{SYM}$ so that its security proof relies directly on the LWE assumption. This new scheme will basically be the same as the previous one, except that the vector a will be chosen according to some biased distribution D_S. The distribution D_S we will use is defined on $\mathbb{Z}_q^n$ and depends on some matrix $S \in \mathbb{Z}_q^{k \times n}$ and a distribution χ.

We will quickly present it in the case where $k = 1$. It verifies the property that $\Pr[D_s = a]$ is proportional to

$$\Pr\left[\left|\left\lfloor \frac{p}{q}.(\overline{\langle a, s\rangle + e})\right\rceil - \frac{p}{q}.(\overline{\langle a, s\rangle + e})\right| < \frac{1}{4}\right],$$

where $e \leftarrow \chi$ and $\overline{\langle a, s\rangle + e}$ means that $\langle a, s\rangle + e$ is interpreted as an element of $\mathbb{Z}$. This basically means that we want the value $(p/q).(\overline{\langle a, s\rangle + e})$ to be close to its rounding for our samples a. One can efficiently sample according to this distribution D_s: sample a uniformly, and output it if and only if, when sampling e according to χ, the value $(p/q).(\overline{\langle a, s\rangle + e})$ is at distance less than $1/4$ from its rounding. Since the distribution of $\langle a, s\rangle + e$ is indistinguishable from uniform, the probability that a vector a gets rejected is (around) $1/2$. To extend this distribution to a matrix S, we will take a distance of $1/2 - 1/4n$ instead of $1/4$.

Definition 15 $(\mathtt{LWE}-\mathtt{SYM})$**.** *Let k and n be polynomials in λ. The symmetric encryption scheme* $\mathtt{LWE}-\mathtt{SYM}$ *is defined in the following way:*

- $\mathsf{Gen}(1^\lambda)$*: output* $S \leftarrow \mathbb{Z}_q^{k \times n}$*;*
- $\mathsf{Enc}_S(x)$*: output* $(a, x + \lfloor a.S \rceil_p)$*, where* $a \leftarrow D_S$*;*
- $\mathsf{Dec}_S(a, y)$*: output* $y - \lfloor a.S \rceil_p$*.*

Our scheme relying on RLWR and MLWR can also be adapted to schemes called $\mathtt{RLWE}-\mathtt{SYM}$ and $\mathtt{MLWE}-\mathtt{SYM}$ in a similar way, that we do not explicit here. We now show that the security of $\mathtt{LWE}-\mathtt{SYM}$ (resp. $\mathtt{RLWE}-\mathtt{SYM}$, $\mathtt{MLWE}-\mathtt{SYM}$) directly reduces to the LWE (resp. RLWE, MLWE) hardness assumption. Our reduction is better than previous ones in the case of one secret. We introduce the problem LWR_D (resp. RLWR_D, MLWR_D) as the same problem as LWR (resp. RLWR, MLWR) except that a is drawn according to the distribution D. To choose secure parameters for LWR, we picked $k = 128$ and $p < \sqrt{q}$ according to [17].

Proposition 3. $\mathtt{LWE}-\mathtt{SYM}$ *(resp.* $\mathtt{RLWE}-\mathtt{SYM}$, $\mathtt{MLWE}-\mathtt{SYM}$*) is semantically secure as soon as the corresponding LWE (resp. RLWE, MLWE) problem is hard.*

Proof. We only prove the standard version, the two other versions being very similar. We will prove our statement by reducing LWR_D to LWE for one secret first, and then explain how to extend it to several secrets. Since $\mathtt{LWE}-\mathtt{SYM}$ basically consists in adding the plaintext to samples from LWR_D, it will prove that our ciphertexts are indistinguishable from uniform.

Consider now an adversary which gets samples from a LWE distribution and has access to an oracle for LWR_D with advantage ϵ. We explicit how it can solve LWE. Given a sample (a, b), the adversary checks whether $\left|\lfloor (p/q).\overline{b} \rceil - (p/q).\overline{b}\right| < 1/4$, and rejects if it is not the case. It collects m samples, and gives them to the LWR_D oracle. Clearly, the set of vectors a which are collected is drawn according to D. Since the probability that a vector a gets rejected can be bounded by a constant, the number of LWE samples needed is a constant times m with high probability.

Now, a sample given to the LWR_D oracle is a true LWR_D sample if and only $\lfloor b \rceil_p = \lfloor \langle a, s \rangle \rceil_p$, where $b = \langle a, s \rangle + e$. If it is not the case, we have $\left| \lfloor b \rceil_p - (p/q).(\overline{\langle a, s \rangle}) \right| > 1/2$, and since all samples satisfy $\left| \lfloor b \rceil_p - (p/q).\overline{b} \right| < 1/4$, we get that $(p/q).e > 1/4$. But, assuming that χ is a Gaussian distribution with standard deviation σ, as in LWE, we have that $\Pr[(p/q).e > 1/4] \leq \exp(-\pi(q/4p\sigma)^2)$. Thus, by taking $q/p > 4\sigma\sqrt{\log(2m/\epsilon)}$, we get that the probability that a sample for our oracle is wrong is less than $\epsilon/2m$. Thus, the probability that all the samples are good is at least $(1 - \epsilon/2m)^m > 1 - \epsilon/2$ (by the union bound). Therefore, LWE is solved with advantage $\epsilon - \epsilon/2$.

Now, in order to extend to a matrix S containing n secret vectors, we assume that the error is a bit smaller, chosen according to a Gaussian with standard deviation $\sigma' = \sigma/n$. We will check whether samples (a, b) verify $\left| \lfloor (p/q).\overline{b_i} \rceil - (p/q).\overline{b_i} \right| < 1/2 - 1/4n$ for all $i \leq n$, which happens with probability $(1 - 1/2n)^n > e^{-\frac{1}{2}}$. Then, any wrong sample will verify that $(p/q).e_i > 1/4n$ for some i and we have $\Pr[(p/q).e_i > 1/4n] \leq \exp(-\pi(q/4np\sigma')^2) = \exp(-\pi(q/4p\sigma)^2)$. Then, we get again that the probability that a sample for our oracle is wrong is less than $\epsilon/2m$, and the result follows similarly. □

Since σ is usually chosen to be at least $\sqrt{k}$ in LWE, our modulus-to-error ratio q/p verifies $q/p > O(n\sqrt{k \log m})$, which is an improvement compared to previous reductions which depend on m rather than $\log m$.

The schemes $\texttt{LWR} - \texttt{SYM}$ and $\texttt{LWE} - \texttt{SYM}$ are similar, except that $\texttt{LWE} - \texttt{SYM}$ involves some checking when generating vectors a. Thus, $\texttt{LWE} - \texttt{SYM}$ has exactly the same efficiency as $\texttt{LWR} - \texttt{SYM}$ for the homomorphic part. The only difference of performance lies in the symmetric encryption, because the generation of the vectors a is a constant factor longer. Thus, we only present the implementation of $\texttt{LWR} - \texttt{SYM}$, because we are only interested in the homomorphic part.

To conclude the presentation of our schemes, we sum up the depth of the decryption circuits in the table 4.4. We count a depth of 0.5 for a multiplication by a constant, consistently with [19]. Our schemes based on LPN require a scalar product with a constant and then decoding. In practice, we will use a 3-repetition code (depth 1), and control the error, requiring 2 additional steps of multiplications, which gives a total depth of 4. For our schemes based on LWR or LWE, there is again a scalar product, followed by some extraction step, which returns the most significant bits of the values.

Scheme	Depth
LPN-C	0.5+decoding depth
LWR-SYM	0.5+extraction step

Table 1. Depths of our schemes

5 Implementing in HElib

HElib is a software library that implements homomorphic encryption. This library is written in C++ and uses the NTL mathematical library. It is based on the BGV FHE scheme introduced by Brakerski, Gentry and Vaikuntanathan in [9].

5.1 Structure of ciphertexts

We describe here the structure of ciphertexts in BGV and HElib, necessary to understand the implementation. The BGV ring-LWE-based scheme is defined over a ring $R =$

$\mathbb{Z}[X]/(\phi_m(X))$, where $\phi_m(X)$ is the m-th cyclotomic polynomial. For an arbitrary integer modulus N, we denote the ring $R_N = R/NR$.

As implemented in HElib, the native plaintext space of the BGV cryptosystem is R_{p^r} for a prime power p^r. The scheme is parametrized by a sequence of decreasing moduli $q_L, \ldots, q_0$, and an "i-th level ciphertext" in the scheme is a vector $c \in (R_{q_i})^2$. Secret keys are elements $s' \in R$ with small coefficients, and we denote $s = (1, s') \in R^2$. A level-i ciphertext $c = (c_0, c_1)$ encrypts a plaintext element $m \in R_{p^r}$ with respect to s if we have $[\langle s, c\rangle]_{q_i} = [c_0 + s \cdot c_1]_{q_i} = m + p^r \cdot e$ in R, for small error term e. This error term grows with homomorphic operations of the cryptosystem, and switching from q_{i+1} to q_i is used to decrease the error term. In practice, HElib provides a routine for finding the smallest value of m allowing L levels of multiplication.

As observed by Smart and Vercauteren [34], an element of the native plaintext space $\alpha \in R_{p^r}$ can be viewed as encoding a vector of "plaintext slots" containing elements from some smaller ring extension of $\mathbb{Z}_{p^r}$ via Chinese remaindering. In this way, a single arithmetic operation on α corresponds to the same operation applied component-wise to all the slots. The number and the size of these slots depends on the factorization of $\phi_m(X)$ modulo p^r (this size is the same for all slots). Addition and multiplications of ciphertexts act on the slots of the corresponding plaintext in parallel.

5.2 Available functionalities

HElib provides the following ciphertext arithmetic functionalities:

- addition, subtraction and multiplication by another ciphertext,
- automorphisms (apply $F(X) \to F(X^k)$),
- addition and multiplication by a constant.

However, these functionalities do not have the same cost in time, as well as in noise. Additions are fast compared to automorphisms or multiplications and operations with constants are fast compared to operations with ciphertexts. Concerning the noise management, multiplications are expensive compared to the other operations, except multiplication with constants whose cost is roughly half the one of a multiplication with a ciphertext. Thus, choices of implementation may have huge impact on the performance and the design of the implementation has to be made specifically for a homomorphic application.

We also used the function ExtractDigits, which performs the extraction of the mod p digits from a mod p^r ciphertext, in our implementation of the $\mathtt{LWR} - \mathtt{SYM}$ scheme. This function has a quite long running time compared to the previously mentioned functionalities.

6 Implementation Description

We have implemented the homomorphic evaluation of the decryption of schemes $\mathtt{LPN} - \mathtt{C}$, $\mathtt{RLPN} - \mathtt{C}$, $\mathtt{MLPN} - \mathtt{C}$, $\mathtt{LWR} - \mathtt{SYM}$ and $\mathtt{RLWR} - \mathtt{SYM}$ using the homomorphic library HElib [23,24], which has already been used in several works about homomorphic evaluations of symmetric schemes [19,28,3].

It is important to understand that moving values which are in a HE ciphertext is costly. Thus, when one wants to do operations on some values, it is better to put them into different HE ciphertexts. He can then pack several similar values in parallel in each ciphertext, and do several times the same computation at once.

For each version of the schemes, we made a bit-sliced implementation, where each digit of the plaintext, ciphertext and key are in different HE ciphertexts. The nature of HElib ciphertexts allows us then to put digits in "slots" and perform several computations in parallel. We also made a "fully-packed" implementation: each slot contains a fixed number of values $(\varphi(m)/nslots)$, number which depends on the parameters. This way, many computations can be done in parallel, leading to a very low running time per bit.

In more details, we had to put the bits of the vector a and of the columns of the matrix S in different HE ciphertexts to compute the scalar product efficiently (and the same for polynomials a and S in ring versions). But packing allowed us to put one row of S in each HE ciphertext. Basically, there is a direct relation between packing and the fact that we can use the same vector a for several secrets in learning problems: we highly use this fact when parallelizing our computations.

In the bit-sliced version, each HE ciphertext contains a row of S, and the resulting symmetric ciphertext y is contained in only one HE ciphertext. In the fully-packed version, several vectors a are generated and encoded together in HE ciphertexts, one by slot, while each row of the symmetric key is encoded in only one slot. Again, the symmetric ciphertext y is contained in one HE ciphertext. Finally, ring versions are quite similar, except that the resulting ciphertext y is bigger and encoded in several HE ciphertexts.

One important optimization is that the symmetric ciphertext does not have to be encrypted as a HE ciphertext, as explained in subsection 3.2. One can just consider it as a constant. This approach reduces the computation time, since multiplication by a constant is less costly than a multiplication between two ciphertexts.

6.1 Implementation of $\mathtt{LPN-C}$

In our implementation of $\mathtt{LPN-C}$, we choose $k = 512$ and $\eta = 10^{-7}$, as suggested in subsection 4.2, as concrete parameters, in order to get 80 bits of security. The blocksize, i.e. the number of secrets we can use for a single random vector a, depends on the HElib parameters. In the bit-sliced version, the block size is bounded by the number of slots, and in the fully-packed version, by the size of the slots.

As explained in subsection 4.2, we used a 3-repetition code, preceded by three iterations of the delinearization steps. These transformations basically consist in multiplying ciphertexts between themself. We generated the indexes x and y used in the transformations randomly (this way, the degree of the values we get is high enough with high probability).

Bit-sliced. As we said, as soon as we want to combine bits together homomorphically, it is more efficient to encrypt them in different HE ciphertexts. Hence, since we basically want to compute a scalar product, we will encrypt each bit of the involved vectors in different HE ciphertexts. So, we have k ciphertexts to represent a and k ciphertexts to represent S. The parallel nature of HElib allows us to encrypt all the columns of S at the same time: each slot will correspond to a column (and will contain one bit). Thus, when computing the scalar product, we will get only one HE ciphertext containing the result, and each slot of this ciphertext will contain one computed scalar product.

Then, we want to decode a 3-repetition code on 3 repetitions of the $\mathtt{LPN-C}$ encryption. We have 3 HE ciphertexts and we compute a majority circuit on them. For each slot, it basically consists in computing the operation $c_1c_2 + c_1c_3 + c_2c_3$. At the end, we get only one HE ciphertext, which contains one bit per slot.

Fully-packed. In the fully-packed version, we take advantage of the fact that each slot of our HE ciphertexts contains elements in $\mathbb{F}_{2^r}$. Thus, instead of having only one bit in each slot, we can put as much as r bits into them. However, the main difficulty is that the values in the slots are considered as polynomials and operations on these values are not made bit-by-bit. There, we use the fact that LPN can be extended to several secrets.

So, we again have k ciphertexts for a and S. But now, each slot of a will contain a bit and each slot of S will contain r bits, where r is the size of the slots, or in other words, the degree of the underlying polynomials. Then, each slot of a is interpreted as a constant polynomial and the multiplication will then correspond to a bit-by-bit multiplication. The result of the scalar product is contained in one HE ciphertext and is also fully-packed, i.e. each slot contains r bits.

However, the computation of the majority circuit on fully-packed HE ciphertexts does not work similarly, because multiplications will not be bit-by-bit. Thus, we use some unpacking technique described in [24]. In a nutshell, it consists in going through a normal basis representation and to consider the applications which give back a coefficient from a polynomial as linear functions. There, any of these linear functions is the composition of one of these linear function with a Frobenius automorphism. Since computing these Frobenius automorphisms is more efficient than directly applying all the linear functions, we improved a bit our performance.

This technique gives us back HE ciphertexts encrypted with only one bit by slot, so that we can compute our majority circuit on them. However, unpacking has a high cost, making the fully-packed version not so efficient.

6.2 Implementation of $\mathtt{LWR-SYM}$

The implementation is very similar to the one of $\mathtt{LPN-C}$, except that no majority circuit is needed. We could have implemented the schemes with binary circuits, representing values in $\mathbb{Z}_q$ bit by bit, but we choose to use the fact that HElib can handle plaintext space modulo m^r: in our case, $m = 2$. Thus, the values modulo p are encoded directly modulo p and not bit by bit. The trickiest part is the rounding operation: since the values contained in the slots are already in $\mathbb{Z}_q$, we have to extract the most significant bits of all the values contained in the ciphertexts.

Fortunately, HElib already contains such a functionality, called ExtractDigits, but it has a huge cost. The function ExtractDigits returns r ciphertexts, where the i-th ciphertext contains the i-th digit of the integers contained in each slot of the input. The way this function works is described in [24].

In our implementation of $\mathtt{LWR-SYM}$, we choose $k = 128$. Based on parameters suggested in [17], we always took $p < \sqrt{q}$. If we do not consider the extraction procedure, then best performances are given for the biggest values of q (which depend on the HElib parameters, a bit less than 2^{32} in general). But the extraction procedure highly depends on q, and we give the best performance for smallest values of q.

6.3 Implementation of ring operations

We implemented polynomial multiplications in two different ways: in a naive way, doing n^2 multiplications for two polynomials of degree n, and via a Fourier transform, leading to a $O(n \log n)$ number of multiplications.

Naive implementation The naive implementation of the ring version is quite similar to the standard version. a and S are still contained in k ciphertexts. The main difference is that, when multiplying a and S, the result is contained in k ciphertexts again. Thus, even if the ring version allows to encrypt more bits at the same time, the fact that the product $a.S$ is encrypted in k ciphertexts leads to a high complexity. Furthermore, the number of multiplications is still n per bit, so the amortized running time is not really improved compared to the standard version.

Fourier tranform We present in Appendix A the Schönhage algorithm [33] we used in order to compute the multiplication of polynomials in fields of characteristic 2 via Fourier transform (for more details, see [30]).

The implementation is similar to the one of the naive algorithm: we implemented standard and ring versions. The main interest is that the number of multiplications for the multiplication via DFT is asymptotically smaller. However, with the concrete parameters we choose, the gain from using Fourier transforms is far from clear, compared to the naive multiplication of polynomials. This is not surprising, as Schönhage algorithm leads to a higher number of homomorphic multiplications with the parameters we choose, in spite of

a better asymptotic complexity. For example, for RLWR with polynomial $X^{128}+1$, instead of 128*128=16384 multiplications in the usual method, we get 27*54*54=78732 multiplications, because we make 27 "small products". In the case of RLPN, instead of 512*512=262144 multiplications, we get 81*54*54=236196 multiplications, which should be a bit more efficient. However, the massive use of homomorphic additions seems to compensate for this gain. We believe that the performance could be improved via Fourier transform, for example by computing small products also via Fourier transform, recursively.

6.4 Implementation of the module version

We implemented the module version, i.e. $\texttt{MLPN}-\texttt{C}$ scheme. The idea of using the module version was indeed an answer to the fact that the majority circuit does not work in the fully-packed version. In the module version, since the multiplications are seen as polynomial multiplications, the operations can be made on fully-packed ciphertexts.

More precisely, d is fixed by the size of the slots. Vectors a and S are encrypted in k ciphertexts, where each slot is full. Then, the result of a scalar product is a fully-packed ciphertext. The majority circuit is directly evaluated on fully-packed ciphertexts.

We remark that our operations are operated with underlying polynomials fixed by HElib. However, since these polynomials are irreducible modulo 2 and there is a homomorphism between all fields $GF(2^d)$, we can pick any of these polynomials, while preserving the security.

7 Performance Analysis

In this section, we provide the performance we got from our implementation and compare to the literature. We run our test on a 4 years-old Mac, with Intel Core i7 running at 2.4GHz.

7.1 HElib parameters and running times

We start by recalling the parameters that we used in HElib. These parameters follow from the number of levels needed by the computation and are directly computed by HElib routines, except L, that we have to provide, but we did several tests to find the smallest accepted value (the number of levels needed by HElib can differ from the theoretical one). They fix the number of slots and values in each slot that HE ciphertexts contain.

	L	m	$\varphi(m)$	nslots
LPN	5	4859	4704	168
LWR	3	4051	4050	81
LWR+extract	18	15709	15004	682

Table 2. HElib parameters: L=number of levels, m=index of the underlying cyclotomic polynomial, $\varphi(m)$=dimension of the underlying ring, nslots=number of slots by ciphertext

We see in Table 2 that there are three kind of sets of parameters: those for LWR which are very low, those for LPN which are a bit higher, and those used for LWR with extraction, which are a lot higher. Indeed, the two steps of multiplications we do in LPN to add noise do not require bigger HElib parameters. Our schemes are a lot more flexible than other schemes from the literature, in the sense that they require drastically lower parameters.

Table 3 gives a summary of the performance of the different schemes we implemented in HElib. We see that the fully-packed version of the LWR scheme is the most efficient. However, one can consider that the form of the output is not processed enough, and this is

Implementations	total running time (s)	time (ms) per bit
LPN − C	28	2.2
LPN − C fully-packed	24*	5*
RLPN − C	1300	15
RLPN − C fully-packed	14500*	6*
MLPN − C	53	0.15
LWR − SYM	0.07	0.1 (8/25)
LWR − SYM + extraction	7.3	5.3 (2/8)
LWR − SYM fully-packed	0.05	0.0008 (16/32)
LWR − SYM fully-packed + extraction	160	5.3 (2/8)
RLWR − SYM	28	0.3 (8/22)
RLWR − SYM + extraction	1100	6.3(2/8)
RLWR − SYM fully-packed	31	0.0075 (8/22)

* Running time without additional multiplication steps

Table 3. Total running time and running time by encoded bit of the homomorphic evaluation of the decryption circuit of our schemes (in parenthesis, for LWR, values of $\log p$ and $\log q$)

why we also provide the running time of the extracted version of the LWR scheme, achieving a total processing of the data.

The schemes based on LWR can have running time down to less than a microsecond per bit. Their total running time is less than a second for standard version and around 25 seconds for ring versions. However, when used with an additional extraction phase, the running time is of the order of few milliseconds per bit and few seconds in total. The LPN-based have running times similar to the one of LWR with extraction. But the best performance is achieved by MLPN − C, with around 0.15 millisecond per bit and less than a minute in total as running time.

For completeness, we also provide in Table 4 the setting performance we got. In the context of the application we presented in subsection 3.2, there is first a (public and offline) phase where the parameters are fixed and the key is homomorphically encrypted. Then, there is a second (online) phase performed by the cloud when receiving the messages, and consisting in the evaluation of the decryption circuit. We do not consider here client side operations, since they are not homomorphic (and thus a lot faster).

Implementations	HElib setting (s)	Key encryption (s)
LPN − C	0.8	9
LPN − C fully-packed	0.7	10
RLPN − C	0.8	10
RLPN − C fully-packed	0.7	12
MLPN − C	0.6	0.4
LWR − SYM	4.5	2
LWR + extraction	32	18
LWR − SYM fully-packed	5.5	1
LWR − SYM fully-packed + extraction	28	19
RLWR − SYM	5.1	2
RLWR + extraction	32	18
RLWR − SYM fully-packed	5	4
RLWR − SYM fully-packed + extraction	28	19

Table 4. Running time of the offline part of the protocol: HElib settings and homomorphic encryption of the symmetric key

Performance of these steps are quite rare in the literature. [19] announces a setting time of around 1.5 minutes. Our setting time is a lot smaller, since our parameters are. The less efficient step in our approach is the homomorphic encryption of the symmetric key, which is usually quite big for lattices, but this step can be done offline, before sending any data. Finally, as we already noticed, the random vector a does not have to be considered as a part of the symmetric ciphertext and does not have to be homomorphically encrypted, which leads to very low online running time.

7.2 Comparison to other works

We give in Table 5 a comparison of the performance of our different schemes with the literature. The symmetric schemes we compare with are `AES` [19] and `LowMC` [3]. We chose to compare with the results of [19,3] because they are the most efficient so far and also implemented in HElib. We do not include the performance of [11] since the ciphers are very slow. In addition, both performance are really recent: `AES` performance has been updated in 2015, and `LowMC` has been presented in 2015. We chose a homomorphic encryption security level of 80 for compatibility with these works.

While both encryption and decryption were performed for `AES`, only encryption performance were given concerning `LowMC` (decryption performance might be slower, as explained in [19]). However, evaluating encryption could also be meaningful, depending on the mode of operation used (for example in counter mode). We believe that the performance of the encryption of our schemes should be very similar to the one of decryption, since the operations which are computed are basically the same.

`LowMC` had so far achieved the best performance for our problem (around 3 milliseconds per bit). However, it is a very recent block cipher, whose security has already been questioned [13,14]. We believe that the fact that our schemes come with a proof is another advantage compared to usual block ciphers.

	total running time (s)	time (ms) per bit
`AES` $-$ `128` encryption[19]	245	16
`AES` $-$ `128` decryption[19]	394	26
`LowMC` [3]	506	3.3
`LPN` $-$ `C`	28	2.2
`MLPN` $-$ `C`	53	**0.15**
`LWR` $-$ `SYM`+extract	**7.3**	5.3
`LWR` $-$ `SYM` fp	0.05	0.0008

Table 5. Comparison of our performance with the literature

One can see that our schemes are a lot more flexible: the total running time of a homomorphic evaluation is roughly one order of magnitude faster. The parameters required by our scheme are very small, because their multiplicative depth is very small. Furthermore, their good performance can extend to bigger parameters: since increasing the size of the parameters allows to pack more bits into the ciphertext, the per bit running time does not grow very fast. As en example, evaluating `LPN` $-$ `C` with a maximum given level of 18 in HElib (which is bigger than for example, the bound used for `LowMC`) instead of 5, i.e. leaving 13 levels of multiplications before bootstrapping gets necessary, leads to a per bite running time of 5.3ms. Thus, we considered that measuring performance with the smallest parameters allowed by our schemes was relevant.

The amortized cost per bit in our scheme based on LWR is extremely low, four orders of magnitude lower than the best cost achieved so far. However, one can consider that the extraction of the bits should be counted in the evaluation time, as computations on the

ciphertext after the evaluation would often need to do this step. We see that even with the additional step of extraction, our scheme is still competitive with the best schemes in the literature. Still, as the extraction step could be included in the bootstrapping that most real-world application would make after the homomorphic evaluation, we consider that our timings without extraction are meaningful.

Finally, as we said, we would suggest to use the scheme $\texttt{MLPN} - \texttt{C}$, which has the shortest amortized running time, has reasonable total running time and returns ciphertexts on which we can directly compute. These results clearly show the interest of using lattice-based symmetric schemes in order to get efficient homomorphic evaluation.

8 Conclusion

Designing FHE-friendly symmetric encryption scheme is an interesting question when trying to make homomorphic encryption practical. Our work provides a new approach, based on lattices, which appears to be the most efficient so far.

Our experiments show that the schemes we presented are competitive with previous propositions. Contrary to other works, which consider block ciphers or stream ciphers, our schemes are provably secure, under some hardness hypotheses. In addition, they are a lot more flexible, in the sense that they require smaller parameters and thus, have drastically lower total running time. We have also shown that encryption schemes with arbitrary shallow decryption circuits are possible, since our LWR scheme has arguably depth zero.

Nonetheless, there is still room for improvement. First, the way the schemes have been implemented in HElib might be optimized, especially the Fourier transform used in the ring versions of the schemes. Parallel computation, which seems a natural assumption for a cloud service, would also lead to drastically faster performance, since the main computation contained in our schemes is a scalar product. Furthermore, one could also design a FHE scheme more "LWR-friendly", which would in particular allow an easy extraction of the most significant bits, which is the main bottleneck to the performance of our LWR scheme in HElib.

References

1. Akavia, A., Bogdanov, A., Guo, S., Kamath, A., Rosen, A.: Candidate weak pseudorandom functions in $AC0 \circ MOD_2$. In: Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014. (2014) 251–260
2. Albrecht, M.R., Cid, C., Faugère, J., Fitzpatrick, R., Perret, L.: Algebraic algorithms for LWE problems. IACR Cryptology ePrint Archive **2014** (2014) 1018
3. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In Oswald, E., Fischlin, M., eds.: EUROCRYPT 2015, Part I. Volume 9056 of LNCS., Springer, Heidelberg (April 2015) 430–454
4. Alwen, J., Krenn, S., Pietrzak, K., Wichs, D.: Learning with rounding, revisited - new reduction, properties and applications. In Canetti, R., Garay, J.A., eds.: CRYPTO 2013, Part I. Volume 8042 of LNCS., Springer, Heidelberg (August 2013) 57–74
5. Applebaum, B.: Cryptographic hardness of random local functions - survey. Electronic Colloquium on Computational Complexity (ECCC) **22** (2015) 27
6. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In Aceto, L., Henzinger, M., Sgall, J., eds.: ICALP 2011, Part I. Volume 6755 of LNCS., Springer, Heidelberg (July 2011) 403–415
7. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In Pointcheval, D., Johansson, T., eds.: EUROCRYPT 2012. Volume 7237 of LNCS., Springer, Heidelberg (April 2012) 719–737
8. Bardet, M., Faugère, J.C., Salvy, B.: On the complexity of gröbner basis computation of semi-regular overdetermined algebraic equations. In: International Conference on Polynomial System Solving - ICPSS. (November 2004) 71 –75

9. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In Goldwasser, S., ed.: ITCS 2012, ACM (January 2012) 309–325
10. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In Ostrovsky, R., ed.: 52nd FOCS, IEEE Computer Society Press (October 2011) 97–106
11. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: How to compress homomorphic ciphertexts. IACR Cryptology ePrint Archive **2015** (2015) 113
12. Cheon, J.H., Coron, J.S., Kim, J., Lee, M.S., Lepoint, T., Tibouchi, M., Yun, A.: Batch fully homomorphic encryption over the integers. In Johansson, T., Nguyen, P.Q., eds.: EUROCRYPT 2013. Volume 7881 of LNCS., Springer, Heidelberg (May 2013) 315–335
13. Dinur, I., Liu, Y., Meier, W., Wang, Q.: Optimized interpolation attacks on lowmc. Cryptology ePrint Archive, Report 2015/418 (2015) `http://eprint.iacr.org/`.
14. Dobraunig, C., Eichlseder, M., Mendel, F.: Higher-order cryptanalysis of lowmc. Cryptology ePrint Archive, Report 2015/407 (2015) `http://eprint.iacr.org/`.
15. Doroz, Y., Hu, Y., Sunar, B.: Homomorphic AES evaluation using NTRU. Cryptology ePrint Archive, Report 2014/039 (2014) `http://eprint.iacr.org/2014/039`.
16. Doröz, Y., Shahverdi, A., Eisenbarth, T., Sunar, B.: Toward practical homomorphic evaluation of block ciphers using prince. In Böhme, R., Brenner, M., Moore, T., Smith, M., eds.: FC 2014 Workshops. Volume 8438 of LNCS., Springer, Heidelberg (March 2014) 208–220
17. Duc, A., Tramèr, F., Vaudenay, S.: Better algorithms for LWE and LWR. In Oswald, E., Fischlin, M., eds.: EUROCRYPT 2015, Part I. Volume 9056 of LNCS., Springer, Heidelberg (April 2015) 173–202
18. Gentry, C.: Fully homomorphic encryption using ideal lattices. In Mitzenmacher, M., ed.: 41st ACM STOC, ACM Press (May / June 2009) 169–178
19. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In Safavi-Naini, R., Canetti, R., eds.: CRYPTO 2012. Volume 7417 of LNCS., Springer, Heidelberg (August 2012) 850–867
20. Gilbert, H., Robshaw, M.J.B., Seurin, Y.: How to encrypt with the LPN problem. In Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I., eds.: ICALP 2008, Part II. Volume 5126 of LNCS., Springer, Heidelberg (July 2008) 679–690
21. Goldwasser, S., Gutfreund, D., Healy, A., Kaufman, T., Rothblum, G.N.: Verifying and decoding in constant depth. In Johnson, D.S., Feige, U., eds.: 39th ACM STOC, ACM Press (June 2007) 440–449
22. Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. In Sarkar, P., Iwata, T., eds.: ASIACRYPT 2014, Part I. Volume 8873 of LNCS., Springer, Heidelberg (December 2014) 1–20
23. Halevi, S., Shoup, V.: Algorithms in HElib. In Garay, J.A., Gennaro, R., eds.: CRYPTO 2014, Part I. Volume 8616 of LNCS., Springer, Heidelberg (August 2014) 554–571
24. Halevi, S., Shoup, V.: Bootstrapping for HElib. In Oswald, E., Fischlin, M., eds.: EUROCRYPT 2015, Part I. Volume 9056 of LNCS., Springer, Heidelberg (April 2015) 641–670
25. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Des. Codes Cryptography **75**(3) (2015) 565–599
26. Lepoint, T., Naehrig, M.: A comparison of the homomorphic encryption schemes FV and YASHE. In Pointcheval, D., Vergnaud, D., eds.: AFRICACRYPT 14. Volume 8469 of LNCS., Springer, Heidelberg (May 2014) 318–335
27. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In Johansson, T., Nguyen, P.Q., eds.: EUROCRYPT 2013. Volume 7881 of LNCS., Springer, Heidelberg (May 2013) 35–54
28. Mella, S., Susella, R.: On the homomorphic computation of symmetric cryptographic primitives. In Stam, M., ed.: IMA Int. Conf. Volume 8308 of Lecture Notes in Computer Science., Springer (2013) 28–44
29. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop. CCSW '11, New York, NY, USA, ACM (2011) 113–124
30. Pospelov, A.: Faster polynomial multiplication via discrete fourier transforms. In: Computer Science - Theory and Applications - 6th International Computer Science Symposium in Russia, CSR 2011. (2011) 91–104
31. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In Gabow, H.N., Fagin, R., eds.: 37th ACM STOC, ACM Press (May 2005) 84–93

32. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Foundations of Secure Computation, Academia Press (1978) 169–179
33. Schönhage, A.: Schnelle multiplikation von polynomen über körpern der charakteristik 2. Acta Inf. **7** (1977) 395–398
34. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Cryptography **71**(1) (2014) 57–81

A Polynomial multiplication in fields of characteristic 2 via Fourier Transform

We briefly present here the Schönhage algorithm [33] we used in order to compute the multiplication of polynomials in fields of characteristic 2 (for more details, see [30]). In the case of LPN, we want to multiply two elements from $\mathbb{Z}_2/(P(X))$, where P is some polynomial irreducible modulo 2. In the case of LWR, the elements are in $\mathbb{Z}_{2^k}/(X^{2^n}+1)$. In both cases, we computed first the multiplication over $\mathbb{Z}_2$ and $\mathbb{Z}_{2^k}$ respectively, before performing the reduction.

Let K be a field of characteristic 2. The usual algorithm for a Fourier transform cannot work there since it requires to have roots of unity of large smooth order. We have to work in an extension of the field, defined by a polynomial which will lead to such roots of unity.

The algorithm reduces the multiplication of two polynomials in K to the multiplication of elements in $A_N = K[x]/(x^{2N}+x^N+1)$, where x is a $3N$-th primitive root of unity. Furthermore, A_N is interpreted as $A_{N_1}[Y]/(Y^{3N_2}+1)$, where $A_{N_1} = K[x]/(x^{2N_1}+x^{N_1}+1)$ and $N = N_1N_2$. The algorithm for multiplying $a, b \in K[X]$ of degree $n-1$ is as follows:

- Choose N a power of 3 such that $N > n$. Then pick $N_1 > N_2$, both powers of 3, such that $N = N_1N_2$.
- Write
$$a = \sum_{i=0}^{N_2-1} \left(\sum_{j=0}^{N_1-1} a_{iN_1+j} x^j \right) (x^{N_1})^i$$
and interpret it as a polynomial over A_{N_1}. Do the same for b. Note that x^{N_1/N_2} is a $3N_2$-th root of unity.
- Do a DFT of length $3N_2$ over A_{N_1} on a and b, using x^{N_1/N_2} as root of unity. Multiplications by powers of x can be performed via additions and cyclic shifts.
- Multiply component-wisely the two vectors of length $3N_2$ (only $2N_2$ multiplications are necessary is the optimized version).
- Do the inverse DFT.
- The j-th coefficient of the i-coefficient in B_{N_1} is added in the (iN_1+j)-th coefficient of the resulting polynomial.

This algorithm performs $3N_2$ multiplications in A_{N_1} ($2N_2$ in the optimized version), because the DFT does not contain any multiplication. The optimized uses some relations into the coefficients to recover a coefficient from two other ones, leading to a 2/3 factor of improvement. The cost of a multiplication in A_{N_1} is at most N_1^2 (it is the cost of the naive algorithm, but it could improved, for example by applying the Fourier transform recursively). Thus the total cost is roughly $2(N/2)^3$.