NFLlib: NTT-Based Fast Lattice Library

Carlos Aguilar-Melchor[1], Joris Barrier[2], Serge Guelton[3], Adrien Guinet[3], Marc-Olivier Killijian[2], and Tancrède Lepoint[4(✉)]

[1] INP-ENSEEIHT, CNRS, IRIT, Université de Toulouse, Toulouse, France
carlos.aguilar@enseeiht.fr
[2] CNRS, LAAS, Université de Toulouse, Toulouse, France
{joris.barrier,marco.killijian}@laas.fr
[3] Quarkslab, Paris, France
{sguelton,aguinet}@quarkslab.com
[4] CryptoExperts, Paris, France
tancrede.lepoint@cryptoexperts.com

Abstract. Recent years have witnessed an increased interest in lattice cryptography. Besides its strong security guarantees, its simplicity and versatility make this powerful theoretical tool a promising competitive alternative to classical cryptographic schemes.

In this paper, we introduce NFLLIB, an efficient and open-source C++ library dedicated to ideal lattice cryptography in the widely-spread polynomial ring $\mathbb{Z}_p[x]/(x^n + 1)$ for n a power of 2. The library combines algorithmic optimizations (Chinese Remainder Theorem, optimized Number Theoretic Transform) together with programming optimization techniques (SSE and AVX2 specializations, C++ expression templates, etc.), and will be fully available under an open source license.

The library compares very favorably to other libraries used in ideal lattice cryptography implementations (namely the generic number theory libraries NTL and FLINT implementing polynomial arithmetic, and the optimized library for lattice homomorphic encryption HELIB): restricting the library to the aforementioned polynomial ring allows to gain several orders of magnitude in efficiency.

Keywords: C++ library · Implementation · Ideal lattice cryptography · Number theoretic transform · Chinese remainder theorem · SEE specializations

Note: NFLlib is available under an open source license at https://github.com/quarkslab/NFLlib

1 Introduction

Lattice cryptography is often praised for its simplicity, its versatility and its possible resistance to quantum attacks. However, its large memory requirements makes its practical use strenuous. The introduction of ideal lattice cryptography

K. Sako (Ed.): CT-RSA 2016, LNCS 9610, pp. 341–356, 2016.
DOI: 10.1007/978-3-319-29485-8_20

completely reshaped this belief [24,27]. In ideal lattice cryptography, primitives rely on the hardness of problems involving polynomial rings in which lattices can be represented by a few polynomials. In recent years, several hardware and software implementations of lattice signatures and encryption have been developed. These implementations show performances competitive with (or even surpassing) those of currently used primitives such as RSA or elliptic curves (see *e.g.* [9,26]). Due to its efficiency and security arguments, ideal lattice cryptography starts to be deployed in products[1] and is promised a bright future.

Besides signature and encryption, lattice cryptography has shown to be amazingly versatile. In particular, most of the homomorphic encryption (HE) schemes rely on lattices. The latter research area is really active, and recent years have seen loads of HE implementations using polynomial rings. Lattices are also used to instantiate schemes with advanced properties, such as identity-based encryption (IBE), functional encryption or multilinear maps.

To work efficiently over polynomials rings in software, we are aware of three main approaches:

(1) Use the *generic* number theory library NTL [30]. This is the approach used in lots of HE implementations (and in particular HElib [16,17]), and in the IBE implementation of [10].
(2) Use the *generic* number theory library FLINT [18]. This is the approach used in [22] to implement two HE schemes, and in [1] for multilinear maps.
(3) Use *home-made* libraries that implement operations in the polynomial ring $\mathbb{Z}_p[x]/(x^n+1)$.
This is the approach used in the open-source VPN implementation [31], Microsoft homomorphic encryption implementation [4], SIMD-optimized implementations [12,15], GPU implementations [7,21] and also [9,26].

Note that *all* the aforementioned implementations consider uniquely (or may be instantiated with) the polynomial ring

$$R_p \stackrel{\text{def}}{=} \mathbb{Z}_p[x]/(x^n+1)$$

for a modulus $p \equiv 1 \pmod{2n}$ and n some power of 2. This setting is *widespread* in ideal lattice cryptography because of its simplicity of exposition and of implementation. Among other advantages, in that setting, polynomials can be multiplied in quasi-linear time using the Number Theoretic Transform (NTT), a Fast Fourier Transform on finite rings [28]. Now, home-made implementations (*i.e.* item (3)) of polynomial operations in the latter setting have shown to achieve better performances than using the generic libraries NTL or FLINT (see *e.g.* [22, Table 4]).[2] This leads us to the following question:

[1] The open-source IPsec-based VPN solution strongSwan [31] includes the BLISS lattice signature [9] as an IKEv2 public key authentication method starting from version 5.2.2.

[2] This is also hinted at in the HElib library [16,17] which modifies the internal routines of NTL to achieve better performances — although for any cyclotomic polynomial ring $\mathbb{Z}_p[x]/(\Phi)$.

How fast can a specialized polynomial library dedicated to lattice cryptography over R_p be?

1.1 Our Contribution: NFLlib

In this work, we present NFLlib, an efficient and scalable `C++` library specialized for cryptography over $R_p = \mathbb{Z}_p[x]/(x^n + 1)$. NFLlib includes optimized subroutines to perform arithmetic operations over polynomials and allows to easily implement ideal lattice cryptography.[3] The library contains algorithmic optimizations (Double-CRT representation, Number-Theoretic Transform, lazy modular reduction), and programming optimizations (Streaming SIMD Extensions, Advanced Vector Extensions, `C++` expression templates).

We benchmarked the library's arithmetic operations over R_p against the generic libraries NTL, FLINT, and against the HE library HElib. Our results show that focusing on a setting widely used in ideal lattice cryptography allowed to gain several orders of magnitude of efficiency compared to generic libraries.

NFLlib will be open-source and available under the GNU General Public License v3.0. It is designed for ideal lattice cryptography, provides a complete set of operations, and minimizes the amount of new code needed to add new features. In short, one of our hopes is that making this library open-source (and thus seeking for contributions) spurs on the development of ideal lattice cryptography in prototypes in the very near future.

1.2 Related Work

Libraries. The NFLlib library is *specialized* for a particular polynomial ring and therefore differs completely from the *generic* libraries NTL [30] and FLINT [18]. These latter libraries allow to perform powerful number theory, while NFLlib focus on a particular polynomial ring. This specialization allowed us to optimize the underlying operations while being designed to be used for ideal lattice cryptography. Another library that implements lattice cryptography is HElib [16,17], which uses NTL. HElib has become a reference to benchmark HE because it implements a full-fledged HE scheme [5] and features efficient packing techniques and other optimizations. Note that NFLlib does not compare to HElib in term of functionality, but NFLlib could replace NTL in HElib when working over R_p, and would yield a much more efficient implementation.

Double-CRT Representation. Using moduli that split completely to store the polynomial coefficients in CRT form (first layer of CRT), and using the NTT representation of the polynomials (second layer of CRT) is a technique that has been used previously in lattice cryptography. In particular, it is used in the

[3] Even though architecture-optimized implementations will always outperform generic libraries, this paper tackles the issue of designing an *efficient* library that can be used on a large range of architecture. Also, NFLlib includes state-of-the-art SSE and AVX2 optimizations for the NTT and the modular multiplication operation.

HElib library [13,16,17] and in the GPU implementation [7]. However, NFLlib features specific primes in the moduli decomposition, chosen to optimize the NTT and allow lazy modular multiplication.

1.3 Outline

In Sect. 2 we describe the basic cryptographic and mathematical notions needed in our paper. In Sect. 3 we present our library, its main components and how these allowed us to get our performance results. In Sect. 4 we compare the performance of our library with other libraries on different algorithms (NTT, multiplications, additions, etc.). Finally, in Sect. 5, we describe some implementations of lattice cryptographic algorithms and highlight the performance results obtained.

2 Preliminaries

Throughout the paper, we let n be a power of 2 and $p > 0$ be a modulus (not necessarily prime as we will see later). Define R to be the ring $R_p \stackrel{\text{def}}{=} \mathbb{Z}_p[x]/(x^n+1)$, *i.e.* the ring of polynomials having integer coefficients, modulo $x^n + 1$. For any integer p, define $R_p \stackrel{\text{def}}{=} \mathbb{Z}_p[x]/(x^n + 1)$ the ring R modulo p, *i.e.* polynomials modulo $x^n + 1$ with coefficients modulo p. We denote by $a \mod b$ the remainder of the euclidean division of a by b, and $c \equiv a \pmod b$ represents any number such that $c \mod b = a$. We use the classical Landau notation.

Ideal Lattice Cryptography. In most of existing implementations, the structured lattices used in ideal lattice cryptography have an interpretation in terms of arithmetic in the ring $\mathbb{Z}_p[x]/(x^n + 1)$, for n a power of 2. Jumping ahead, note that we will chose particular values for p in order to optimize the polynomial multiplications when using the Number Theoretic Transform (see also [9,15,29]) in combination with the Chinese Remainder Theorem.

The Chinese Remainder Theorem (CRT). Throughout the paper, the modulus p will be composite and square-free, and its factorization is denoted $p = p_1 \cdots p_\ell$. The CRT yields an isomorphism $\mathbb{Z}_p \simeq \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_\ell}$, which extends directly to polynomials rings. In particular, we have that $R_p \simeq R_{p_1} \times \cdots \times R_{p_\ell}$. Jumping ahead of Sect. 3.1, the latter equivalence shows that, instead of working with a polynomial $a(x) \in R_p$, we will choose $p = p_1 \cdots p_\ell$ with particular p_i's and work with ℓ polynomials $a_i(x) \in R_{p_i}$.

The Number Theoretic Transform (NTT). To multiply polynomials efficiently, we use the quasi-linear polynomial multiplication algorithm called the NTT [28]. The advantages of using NTT for ideal lattice cryptography were recently demonstrated in hardware and software implementations [9,14,15,29].

3 NFLlib: A Library for Ideal-Lattice Cryptography

In this section, we introduce NFLlib, a `C++` library for ideal-lattice cryptography, *i.e.* for manipulating polynomials of $R_p = \mathbb{Z}_p[x]/(x^n + 1)$. The entry point to our library is a templated class `poly < class T, size_t degree, size_t sizeModulus >`.

To obtain a usable class, one must define three parameters before compilation: the word type (`uint16_t`, `uint32_t` or `uint64_t`), the degree n of the polynomial $x^n + 1$ that defines R_p (which must be a power of two), and the bit-size of the modulus p, which will be internally constructed as a product of ℓ fixed-size primes: $p = p_1 \times \cdots \times p_\ell$.

The `poly` class features: overloaded operators for modular arithmetic and data manipulation (and the associated static functions); `C++` template expressions to minimize the inherent performance degradation of overloaded operator; functions to sample polynomials in R_p with different distributions for the coefficients (uniform distribution modulo p, uniformly bounded distribution, discrete Gaussian distribution); transformation-related functions (NTT, CRT, export, import); SSE and AVX2 optimizations for compatible architectures.

The word type `T` is the most critical parameter. It defines which (and how many) primes p_i's are available, what is the maximal polynomial degree n possible, and which underlying code is used. Indeed, the code is specialized and might feature SIMD optimizations (especially when using 32-bit and 16-bit words).

All the functions provided by the `poly` class have been developed from scratch, and are based on the native (scalar or vectorial) instructions of a modern CPU. Only exceptions, the Salsa20-based pseudo-random number generator, and the CRT inversion function which uses GMP if the modulus used is too large for native instructions.

NFLlib is a specialized polynomial library dedicated to ideal-lattice cryptography. It is well known that in this setting representing polynomials by their values instead of their coefficients (*i.e.* representing them in NTT form) and using the CRT to represent values is very beneficial for performance. We therefore use such a representation.

NFLlib's performance results are mainly due to the fact that most of the functions have been developed directly based on native operations, and to four major choices that have proven to be very efficient. These choices are:

- the fixed-size CRT representation — see Sect. 3.1;
- the modular multiplication for scalars — see Sect. 3.2;
- the NTT algorithm — see Sect. 3.3.
- the SSE and AVX2 optimizations — see Sect. 4.

In the aforementioned sections, we describe the particular choices we made in NFLlib, and discuss their respective impacts.

3.1 Fixed-Size CRT Representation

For efficiency reasons, we selected the moduli p used in our ideal lattice setting as a product of ℓ fixed-size primes p_i's fitting on one word.[4] Thus, one can work with the CRT representation $(a_1(x), \ldots, a_\ell(x)) \in R_{p_1} \times \cdots \times R_{p_\ell}$ of a polynomial $a(x) \in R_p$, where $a_i(x) = a(x) \mod p_i$. All the $a_i(x)$ can then be processed independently.

The primes forming the moduli are chosen with the following constraints:

Constraint 1. Their size must be at most the word size minus two bits, so that we can do lazy modular reductions in the NTT algorithm (which gives roughly a 30 % speedup);

Constraint 2. They must satisfy Eq. (1) for a given parameter s_0, in order to use the modular multiplication algorithm of Sect. 3.2;

Constraint 3. For any possible value of n — the degree of the quotient polynomial in R_p — they must be congruent to 1 (mod $2n$), so that we can find n-th roots of -1 to use the NTT algorithm of Sect. 3.3 and do polynomial multiplications modulo $x^n + 1$.

Constraint 2 will ensure that Constraint 1 is satisfied when $s_0 \geq 2$. By default, NFLLIB sets $s_0 = 2$. In order to satisfy Constraint 3, we had to arbitrarily select a maximal polynomial degree $n_{\max}$ in NFLLIB. (Note that the constraint is then satisfied for any degree $n \leqslant n_{\max}$). The higher $n_{\max}$ is, the less primes verify Constraint 3. When the word size is 16 bits, these constraints are stronger than for larger words. For example for $n_{\max} = 2048$, only one 14-bit prime verifies Constraint 3 (supposing $s_0 = 2$). For 64-bit words on the other hand, it is possible to find thousands of primes verifying the constraints even for very large polynomial degrees such as $n_{\max} = 2^{20}$. Algorithm 1 returns the primes satisfying Contraints 1–3.

Defining these primes statically is beneficial for performance, and therefore they have been included in a parameter file `params.hpp` with $n_{\max} = 512$ when $s = 16$ (2 primes), $n_{\max} = 2^{15}$ when $s = 32$ (291 primes), and $n_{\max} = 2^{20}$ when $s = 64$ (primes limited voluntarily to one thousand). All of these have been chosen with $s_0 = 2$ as explained before. Of course other values of s_0 and $n_{\max}$ may be defined by the user of NFLLIB.[5]

[4] At the heart of many kinds of ideal-lattice schemes (ranging from classical encryption to fully homomorphic encryption and multilinear maps) is the decision-Ring-Learning-With-Errors (dRLWE) assumption. Working with cyclotomic polynomials $\Phi(x) = x^n + 1$ implies that we have *provable worst-case hardness* for dRLWE with essentially **any** large enough p — splitting, inert, or anywhere in between [6]. In NFLLIB, we therefore chose a p that splits completely for efficiency reasons.

[5] NFLLIB has been designed to work with a wide range of parameters: polynomial degrees $2 \leqslant n \leqslant 2^{20}$ and moduli $2^{13} < p < 2^{1000 \cdot 62}$. However, the users of NFLLIB are responsible for selecting parameters (n, p) that ensure κ bits of security for the specific application they are developing. We refer to [2,3,23] for selecting concrete security parameters of lattice encryption schemes.

Algorithm 1. Prime selection algorithm

Input: s word size, s_0 margin bits, $n_{\max}$ maximum polynomial degree
Output: $(p_1, \ldots, p_t)$ a list of primes satisfying Constraints 1-3

$\beta = 2^s, i = 1,$ `outputList` $= ()$
do
  $c = \beta/2^{s_0} - i \cdot 2^{n_{\max}+1} + 1$
  if `isPrime`(c) *and* $c > (1 + 1/2^{3s_0}) \cdot \beta/(2^{s_0} + 1)$ **then**
    Add c to `outputList`
  end
  i = i+1
while $c > (1 + 1/2^{3s_0}) \cdot \beta/(2^{s_0} + 1)$
return `outputList`

3.2 Optimizing the Modular Multiplication

As explained in Sect. 3.1, NFLLIB includes invariant primes of 14, 30 and 62 bits, and computations are performed independently over these primes. However — as already emphasized in [25] — computing modular reductions with an invariant integer using a well-tuned Newton reciprocal followed by multiplications and adjustments wins over the hardware division instructions.

During the library construction, we observed that the `gcc` compiler automatically optimized the modular multiplications when working with 16-bit or 32-bit words (*i.e.* for 14- and 30-bit primes), but not with 64-bit words. In this section, we consider the problem of dividing a two-word integer by a single word integer. This problem was extensively studied in [25] which proposed a new algorithm (Algorithm 4 in the latter paper) giving a speedup of roughly 30 % over the Newton reciprocal algorithm [25, Algorithm 1]. The former algorithm was included in the version 4.3 of the GMP library.

However, in NFLLIB, the primes are feature so that their (two) most significant bits equal to 0, and the algorithms in [25] are optimized for numbers with their most significant bit equal to 1. In the rest of the section, we describe a new algorithm which significantly improves over [25] for numbers p smaller than the word base $\beta = 2^s$, as illustrated in Table 1.

Table 1. Time per componentwise multiplication of polynomials of degree 1024 modulo a 62-bit prime (average over 100,000 polynomial multiplications on an Intel Xeon CPU E5-2666 v3 at 2.90GHz). We implemented Algorithms 1 and 4 of [25] (with $4p$ instead of p and two conditional subtractions at the end), but they perform one order of magnitude slower than our improved algorithm.

Algorithm	Naive (*i.e.* using %)	[25] Algorithm 1	[25] Algorithm 4	Ours Algorithm 2
Polynomial Modular Mult. (µs)	29.8 µs	15.5 µs	12.9 µs	**2.90 µs**

Algorithm 2. Modular reduction with a modulus verifying Eq. (1)

Input: $u = \langle u_1, u_0 \rangle \in [0, p^2)$, p verifying Eq. (1), $v_0 = \lfloor \beta^2/p \rfloor \mod \beta$, $1 \leqslant s_0 \leqslant s-1$ margin bits
Output: $r = u \mod p$
1 $q \leftarrow v_0 \cdot u_1 + 2^{s_0} \cdot u \mod \beta^2$
2 $r \leftarrow u - \lfloor q/\beta \rfloor \cdot p \mod \beta$
3 **if** $r \geqslant p$ **then** $r \leftarrow r - p$
4 **return** r

Assume that one wants to compute a modular reduction with a modulus p such that

$$(1 + 1/2^{3s_0}) \cdot \beta/(2^{s_0} + 1) \ < \ p \ < \ \beta/2^{s_0}\,, \tag{1}$$

for an integer $1 \leqslant s_0 \leqslant s-1$ (note that all our 62-bit primes verify Eq. (1)). For any number $u \in [0, \beta^2)$, denote $\langle u_1, u_0 \rangle$ its decomposition in words smaller than β, so that $u = u_1 \cdot \beta + u_0$. We describe our new modular reduction in Algorithm 2.

We have the following theorem. For space constraints, we defer its proof to the final version of the paper.

Theorem 1. *Assume* $1 \leqslant s_0 \leqslant s-1$ *and* p *verifies Eq.* (1) *and* $u = \langle u_1, u_0 \rangle \in [0, p^2)$. *Let* $v = \langle v_1, v_0 \rangle = \lfloor \beta^2/p \rfloor$. *Then Algorithm 2 with input* (u, p, v_0, s_0) *outputs* $(u \mod p)$.

3.3 A Lazy NTT Algorithm

We use Harvey's NTT algorithm [14]. This algorithm uses two techniques to reduce its computational costs: pre-computed quotients to accelerate modular multiplications, and lazy reductions (*i.e.* inputs and outputs can be up to twice the modulus). Quotient pre-computations in the NTT was already performed by NTL [30] but Harvey proves elegantly that the NTT butterflies can be modified so that the output is in $[0, 2p)$ when the input is in $[0, p)$, using only one conditional subtraction (instead of three in the initial algorithm). This gives a very nice performance boost of about 30 %, as shown in [14]. Note that this justifies to select primes ad in Sect. 3.2.

As usual, before applying the NTT we multiply the i-th coordinate of the polynomial we are going to transform by ψ^i, where ψ is an n-th root of -1 which allows us to have negatively wrapped convolutions when we multiply two elements (*i.e.* reductions modulo $x^n + 1$). After the NTT, we reduce the coefficients to $[0, p)$ but we do not apply the bit-reverse permutation by default. The reason for this is that, in lattice based cryptography, we often want to offload work from the NTT to the inverse NTT. For example in an LWE encryption scheme, at encryption time one needs to: (1) generate multiple noise polynomials, (2) convert each of them with an NTT, and (3) multiply/add them. In the decryption phase, on the other hand, there is no noise polynomials to generate

and there is just one multiplication, one addition and a single inverse NTT. If in a given case, such as the one described in Sect. 5.1 we want to balance both transformations, such a change can be activated with a compilation option.

Our library has no particular contribution concerning the NTT, we just show in this paper that it is a lot more efficient than the Bluestein FFT used in HElib (see Sect. 4). Our implementation does not use assembly language, but it is quite efficient, scalable and general.

4 Performances Evaluation and Comparison with NTL, FLINT and HElib

In this section we analyze the performance of our library and report comparative benchmarks with the NTL [30], FLINT [18] and HELIB [16] libraries.

Recall that NTL and FLINT are generic libraries that allow to work with polynomials in any modular rings, and HELIB is a software library (based on NTL) that implement an optimized version of the Brakerski-Gentry-Vaikuntanathan [5] (BGV) homomorphic encryption scheme. We chose to compare to these libraries because they are widely used in the literature on lattice cryptography implementations. We restricted them to the same settings as NFLLIB, *i.e.* to work over $R_p = \mathbb{Z}_p[x]/(x^n + 1)$ with moduli p as in Sect. 3.1.

Setting. We benchmarked NFLLIB against NTL, FLINT and HELIB on random polynomial generation, NTT and inverse NTT, modular addition and multiplication in NTT representation. All the benchmarks were made using the following fixed parameter sets:

(1) $n = 256$ with a modulus p of 14 bits,
(2) $n = 512$ with a modulus p of 30 bits,
(3) $n = 1024$ with a modulus p of 62 bits,
(4) $n = 1024$ with a modulus p of about 6200 bits (product of 100 62-bit moduli).

As expected, NFLLIB has been instantiated with 16-bit words and 32-bit words respectively for the parameters sets 1 and 2. For NTL, we used the `zz_pX` objects for the parameters sets 1 and 2, and `ZZ_pX` otherwise. For FLINT, we used the type `fmpz_mod_poly_t`. Finally, HELIB includes a `DoubleCRT` class with the same representation as NFLLIB.[6]

We performed all our benchmarks on a `c4.2xlarge` instance of Amazon Web Services with an Intel Xeon CPU E5-2666 v3 (Haswell) at 2900 Mhz and 15 GB of RAM with `gcc 4.9`, GMP 6.0, NTL 8.1, FLINT 2.5.[7]

[6] In HELIB, the instantiation of a `FHEContext` — storing the modulus decomposition — is needed to use `DoubleCRT` objects. Now, this constructor try to produced primes of a size close to 44 bits and this size is hard-coded in the value `FHE_p2Size` (maybe to fit largely the `long` primitive type and be able to do specific homomorphic operations?).

For the sake of comparison, we kept this hardcoded value. Therefore the benchmarks of HELIB are with a 44-bit prime for parameters (1) and (2), with two 44-bit primes for parameters (3) and 141 44-bit primes for parameters (4).

[7] TurboBoost and Hyperthreading were disabled during the benchmarks. We chose an AWS machine as a typical cloud environment which allows reproductibility of the results.

Remark 1. To demonstrate the performance of our library on different architectures, we also benchmarked the NTT transformation on a MacBook Air (called `macbookair`) with an Intel Core i7-4650U Processor at 1700 Mhz and 8 GB of RAM, using the native `clang++` (Apple LLVM version 6.0), GMP 6.0, NTL 8.1, FLINT 2.5. (We restricted ourselves to the benchmark of the NTT transform to be concise.)

Random Polynomial Generation. To benchmark random generation, we used the `ntl::random` function of NTL, the `fmpz_mod_poly_randtest` function of FLINT and the default random generator of NFLLIB (described only in the full version due to space constraints). We present our results in Table 2. Note that the FLINT library implements the Mersenne Twister algorithm that is unsuitable for a cryptographic use.

Table 2. Timings to generate random polynomials in $\mathbb{Z}_p[x]/(x^n+1)$ using the built-in functions of different libraries on `c4.2xlarge`.

Library	NTL `random`	FLINT `fmpz_mod_poly_randtest`	HELIB	NFLLIB `nfl::uniform`
$(1) = (256, 14)$	9.2 µs	4.8 µs	69 µs	**0.6 µs**
$(2) = (512, 30)$	23.2 µs	9.1 µs	135.5 µs	**2.6 µs**
$(3) = (1024, 62)$	173.0 µs	18.3 µs	540.0 µs	**9.7 µs**
$(4) = (1024, 6200)$	8675 µs	1082 µs	37929 µs	**1029.6 µs**

NTT and iNTT. Working with the NTT representation of polynomials (after the negative wrapped convolution) is a cornerstone of our optimization, since additions and multiplications become essentially linear in the number of coefficients. We report in Table 3 the benchmarks of the NTT (*including* the negative wrapped convolution). Note that NTL provides an NTT functions thanks to `TofftRep` and to `toFFTRep` (resp. for `zz_pX` and `ZZ_pX`); no such functions seem to be available in the FLINT library.[8] In HELIB, the `DoubleCRT` class has two functions to convert from (via negative wrapped convolution and NTT) and to (via inverse NTT and inverse of the convolution) a polynomial `ZZX`. (For space constraints, the timings of the inverse NTT are provided in the full version of the paper).

SEE and AVX2 Optimizations. Because of the highly parallel nature of operations over polynomials (the same operations are to be performed on multiple data objects), using Streaming SIMD Extensions (SSE) and Advanced Vector Extensions (AVX) instructions might greatly increase performance. This has been shown in [12,15] respectively for lattice signature and encryption.

[8] We neglected the cost of the (linear) negative wrapped convolution computation in NTL to mitigate the impact of a non highly-optimized hand-made implementation; one would therefore have to expect slightly worse timings when working over R_p.

Table 3. Timings to compute the Number Theoretic Transform of a polynomial in $\mathbb{Z}_p[x]/(x^n+1)$ using (when possible) the built-in functions of different libraries.

(a) NTT on `c4.2xlarge` using `gcc`				
Library	NTL `toFFT/ToFFT`	FLINT	HElib `conv(DoubleCRT,ZZX)`	NFLlib
$(1) = (256, 14)$	7.2 µs	–	33.7 µs	**2.5 µs**
$(2) = (512, 30)$	14.7 µs	–	70.7 µs	**4.5 µs**
$(3) = (1024, 62)$	45.7 µs	–	317.7 µs	**13.9 µs**
$(4) = (1024, 6200)$	33921 µs	–	23240 µs	**1341.0 µs**
(b) NTT on `macbookair` using `clang`				
Library	NTL	FLINT	HElib	NFLlib
$(1) = (256, 14)$	7.7 µs	–	37.6 µs	**1.7 µs**
$(2) = (512, 30)$	16.0 µs	–	74.9 µs	**5.7 µs**
$(3) = (1024, 62)$	47.5 µs	–	333.8 µs	**15.3 µs**
$(4) = (1024, 6200)$	34799 µs	–	24713 µs	**1163.4 µs**

NFLlib includes SSE and AVX2 specializations of the NTT algorithm and of the modular operations for 16-bit and 32-bit words. We compared NFLlib's NTT to Güneysu *et al.* AVX-optimized NTT [15] (where once again the NTT includes the negative wrapped convolution Ψ) on `c4.2xlarge`.

The GOPS implementation works with the `double` type for a 23-bit modulus p (lazy-reduction) and takes 5030 cycles. NFLlib can be instantiated with 14-bit primes or 30-bit primes and takes respectively 3324 and 7334 cycles when using SSE4 instructions, and 2767 and 5956 cycles when using AVX2 instructions. As a comparison, the 62-bit version (i.e. non-SIMD) of the NTT takes 10020 cycles.

5 Implementing Ideal Lattice Cryptography with NFLlib

5.1 High Performance Key Exchange

In this section, we consider an equivalent of the key transport protocol RSASVE of NIST SP 800 56B, using [23] encryption scheme, to illustrate the performances of our library in a concrete setting. The client chooses a random message and encrypts it with the server public key then, the server decrypts this random value that is used to derivate (with a hashing function) a common secret.

Server-Side Focus. As a server usually has to handle many clients, the main issue is how costly is the server-side computation. Thus, we focus on the server cost.

Server Authentication and Forward Secrecy. The public key sent by the server may be a certificate signed by any algorithm (*e.g.* DSA) so that the client is able to be convinced of the server's identity. Since this has no cost for the server we do

not focus on which signature scheme is used. We note that as suggested in [20], the server can send two keys: one signed to prove his identity, and one ephemeral key generated to ensure forward secrecy. Then the client sends two secrets and the common secret is derived from both initial secrets with a key derivation function (*e.g.* a hash function). Due to the signature of one of the public keys, the client knows that only the server can get the common secret and if the ephemeral key is destroyed at the end of the key exchange, forward secrecy is ensured. This means that from the server side multiplying the communication and computational costs just by two, allows to have a forward secrecy property.

The algorithm we implemented is the RLWE encryption scheme of [23].[9] The code for the encryption and decryption functions (see [23]) is presented in Algorithms 3 and 4. This code highlights how simple is to implement algorithms with NFLLIB: the encryption function and decryption functions are very readable, and have respectively 9 and 4 lines of code.

Algorithm 3. Ring-LWE based public key encryption function

Input: `P` a polynomial type, `g_prng` Gaussian generator, `pka`, `pkb` public key, `m` the message

Output: `resa`, `resb` an encryption of `m`

```
using value_t = typename P::value_type;
P tmpu = nfl::gaussian<value_t>(g_prng);     // no noise multiplier
P tmpe1 = nfl::gaussian<value_t>(g_prng, 2); // noise multiplier: 2
P tmpe2 = nfl::gaussian<value_t>(g_prng, 2); // noise multiplier: 2
tmpe2 += m;

tmpu.ntt_pow_phi();
tmpe1.ntt_pow_phi();
tmpe2.ntt_pow_phi();

resa = tmpu * pka + tmpe1;
resb = tmpu * pkb + tmpe2;
```

Table 4 shows the performances of the protocols for 80, 128 and 256 bits of security. In RSA and NFLLIB, the server needs to do a decryption, while in ECDH it performs a modular exponentiation. NFLLIB allows to deal with more clients or to use less CPU time for the same amount of clients. The gap is around a factor 200, so it is possible to process 10 times more clients with 10 times less CPU time and to increase by a factor two the security with respect to ECDH (or maintain the security level and add forward secrecy).

[9] We choose two parameter sets from [23], a 14-bit modulus with polynomials of degree 256, and the same modulus with polynomials of degree 512. These two parameter sets correspond roughly to 128 and 256 bits of security. Note that if these estimates are too low it is possible to choose parameters such as $(14, 1024)$ and the performance presented in Table 4 is just divided by two.

Algorithm 4. Ring-LWE based public key decryption function

Input: `P` a polynomial type, `resa`, `resb` a ciphertext, `s` a secret key, p a modulus
Output: A polynomial `m`

```
m = resb - resa * s;
m.invntt_pow_invphi();
for(auto & v : m)
  v = (v<modulus/2) ? v%2 : 1-v%2;
```

Table 4. Number of key exchanges per second on a server with an i7-4770 processor using only one core. When the four cores are used, performance are multiplied by a factor four. There is no standard implementation of RSA15360 and our library does not work with 80 bits of security for this application (hence the input N/A). RSA and ECDH (p curves) results have been obtained with the speed test of openssl 1.0.1f. The results noted NFLlib correspond to the amount of decryptions per second with our implementation of the RLWE scheme of [23].

Protocol	80 bits	128 bits	256 bits
RSA	7.95 Kops/s	0.31 Kops/s	N/A
ECDH	7.01 Kops/s	5.93 Kops/s	1.61 Kops/s
NFLlib	N/A	1020 Kops/s	508 Kops/s

5.2 Using NFLlib for Homomorphic Encryption

A trending application of ideal lattice cryptography is homomorphic encryption; a fully homomorphic encryption (FHE) scheme enables one to process any function on encrypted data. The first implementations of FHE were quite inefficient, but in six years the landscape has considerably changed and recent implementations run in reasonable time [11,17]. However, the bootstrapping procedure — necessary to achieve *fully* homomorphic encryption — remains a bottleneck.

To overcome thereof, the cryptographic community focused on somewhat homomorphic encryption (SHE) schemes, *i.e.* schemes only able to handle a *bounded* number of homomorphic operations (and especially of homomorphic multiplications). However, even for this simplified setting, to homomorphically evaluate non trivial functions the parameter sizes remain very large (see *e.g.* [8, 22]); to handle around 40 levels, one usually works with parameters such that $2^{10} \leqslant n, \log q \leqslant 2^{20}$.

These large parameters explain why the static parameters in NFLlib were selected to handle polynomials up to degree 2^{20} and modulus up to $62,000$ bits. Now, from the results of Sect. 4, we estimate that implementations using NTL or flint with R_p should immediately gain a factor 15 to 50 in performances by using NFLlib. As an example, we modified the open-source implementation of the somewhat homomorphic encryption scheme FV of [22] and directly replaced flint by NFLlib— we obtained the improvements described in Table 5.

Table 5. Using NFLLIB in the FV implementation of [22], instead of FLINT. The polynomial degree is $n = 4096$ and the modulus p has 124 bits. The relatively small gain on the homomorphic multiplication can be explained by the fact that the scale-invariant procedure is essentially constituted of operations independent of NFLLIB, such as divisions and rounding.

	`Encrypt`	`Decrypt`	`Hom. Add.`	`Hom. Mult.`
[22] with FLINT	26.7 ms	13.3 ms	1.1 ms	91.2 ms
[22] with NFLLIB	**0.9 ms**	**0.9 ms**	**0.01 ms**	**17.2 ms**
Gain	×30	×15	×110	×5.5

6 Conclusion

This work introduces NFLLIB, an optimized open-source `C++` library designed to handle polynomials over $\mathbb{Z}_p[x]/(x^n+1)$, a widespread setting in ideal lattice cryptography. Because of its algorithmic and programming optimizations, NFLLIB is much faster than NTL and FLINT, and as fast as AVX-optimized implementations of the literature. We hope the library will help building efficient prototypes using lattice cryptography in the very near future.

Acknowledgements. This work has been supported in part by the European Union's H2020 Programme under grant agreement number ICT-644209 and by French's FUI project CRYPTOCOMP.

References

1. Albrecht, M.R., Cocis, C., Laguillaumie, F., Langlois, A.: Implementing candidate graded encoding schemes from ideal lattices. Cryptology ePrint Archive, Report 2014/928 (2014). http://eprint.iacr.org/2014/928
2. Albrecht, M.R., Fitzpatrick, R., Göpfert, F.: On the efficacy of solving LWE by reduction to unique-SVP. In: Lee, H.-S., Han, D.-G. (eds.) ICISC 2013. LNCS, vol. 8565, pp. 293–310. Springer, Heidelberg (2014)
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046 (2015). http://eprint.iacr.org/2015/046
4. Bos, J.W., Lauter, K., Loftus, J., Naehrig, M.: Improved security for a ring-based fully homomorphic encryption scheme. In: Stam, M. (ed.) IMACC 2013. LNCS, vol. 8308, pp. 45–64. Springer, Heidelberg (2013)
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012, pp. 309–325. ACM, January 2012
6. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC, pp. 575–584. ACM Press, June 2013

7. Dai, W., Doröz, Y., Sunar, B.: Accelerating NTRU based homomorphic encryption using GPUs. In: IEEE High Performance Extreme Computing Conference, HPEC 2014, Waltham, MA, USA, 9–11 September 2014, pp. 1–6. IEEE (2014)
8. 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. LNCS, vol. 8438, pp. 208–220. Springer, Heidelberg (2014)
9. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013)
10. Ducas, L., Lyubashevsky, V., Prest, T.: Efficient identity-based encryption over NTRU lattices. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 22–41. Springer, Heidelberg (2014)
11. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 617–640. Springer, Heidelberg (2015)
12. El Bansarkhani, R., Buchmann, J.: High performance lattice-based CCA-secure encryption. Cryptology ePrint Archive, Report 2015/042 (2015). http://eprint.iacr.org/2015/042
13. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 850–867. Springer, Heidelberg (2012)
14. Göttert, N., Feller, T., Schneider, M., Buchmann, J., Huss, S.: On the design of hardware building blocks for modern lattice-based encryption schemes. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 512–529. Springer, Heidelberg (2012)
15. Güneysu, T., Oder, T., Pöppelmann, T., Schwabe, P.: Software speed records for lattice-based signatures. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 67–82. Springer, Heidelberg (2013)
16. Halevi, S., Shoup, V.: Algorithms in HElib. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 554–571. Springer, Heidelberg (2014)
17. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 641–670. Springer, Heidelberg (2015)
18. Hart, W., et al.: Fast library for number theory (Version 2.5) (2015). http://www.flintlib.org
19. Harvey, D.: Faster arithmetic for number-theoretic transforms. J. Symb. Comput. **60**, 113–119 (2014)
20. Itkis, G.: Forward security, adaptive cryptography: time evolution (2004). http://www.cs.bu.edu/fac/itkis/pap/forward-secure-survey.pdf
21. Khedr, A., Gulak, G., Vaikuntanathan, V.: SHIELD: scalable homomorphic implementation of encrypted data-classifiers. Cryptology ePrint Archive, Report 2014/838 (2014). http://eprint.iacr.org/2014/838
22. Lepoint, T., Naehrig, M.: A comparison of the homomorphic encryption schemes FV and YASHE. In: Pointcheval, D., Vergnaud, D. (eds.) AFRICACRYPT. LNCS, vol. 8469, pp. 318–335. Springer, Heidelberg (2014)
23. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (2011)
24. Lyubashevsky, V., Micciancio, D.: Generalized compact knapsacks are collision resistant. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 144–155. Springer, Heidelberg (2006)

25. Moller, N., Granlund, T.: Improved division by invariant integers. IEEE Trans. Comput. **60**(2), 165–175 (2011)
26. Oder, T., Pöppelmann, T., Güneysu, T.: Beyond ECDSA and RSA: lattice-based digital signatures on constrained devices. In: The 51st Annual Design Automation Conference 2014, DAC 2014, San Francisco, CA, USA, 1–5 June 2014, pp. 1–6 (2014)
27. Peikert, C., Rosen, A.: Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 145–166. Springer, Heidelberg (2006)
28. Pollard, J.M.: The fast Fourier transform in a finite field. Math. Comput. **25**(114), 365–374 (1971)
29. Pöppelmann, T., Güneysu, T.: Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In: Hevia, A., Neven, G. (eds.) LatinCrypt 2012. LNCS, vol. 7533, pp. 139–158. Springer, Heidelberg (2012)
30. Shoup, V.: Number theory library (Version 8.1) (2015). http://www.shoup.net/ntl
31. Steffen, A., et al.: strongSwan (Version 5.2.2) (2015). https://www.strongswan.org/