Journal of Theoretical and Applied Information Technology
15th March 2021. Vol.99. No 5

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN EFFICIENT FULLY HOMOMORPHIC ENCRYPTION SCHEME OVER NTRU_ROBUST_PKE

[1]EL HASSANE LAAJI, [1]ABDELMALEK AZIZI, AND [2]TAOUFIK SERRAJ
[1]ACSA Laboratory , Mohammed First Uniersity, Oujda, Morocco
[2]MASI Laboratory, Multidisciplinary Faculty, Nador, Morocco
e.laaji@ump.ac.ma, abdelmalekazizi@yahoo.fr, taoufik.serraj@gmail.com

ABSTRACT

Fully Homomorphic Encryption(FHE) is an important field of research. This modern technology enables computation over encrypted data by a cryptosystem able to perform correctly those calculations. In this work, we create an improved release of NTRU called NTRUrobus_PKE (Public Key Encryption), and we create over it an efficient Fully Homomorphic Encryption (FHE) scheme. Our NTRUrobust_PKE is implemented by using the Number Theoretic Transform (NTT) algorithm combined with our own Fast Modular Multiplication algorithm (FMMA). Using these algorithms allows the complete cryptographic process to be faster by a factor up to 69 times compared to using just the convolution multiplication instead. And that allows our FHE protocol over NTRUrobust_PKE to perform iteratively the computation over the encrypted data with perfect correctness and for unlimited depth. In terms of security, we target the high level by using the parameters set that meets the category 5 security level defined by NIST (National Institute for Standard and Technology).
Keywords: *Post Quantum cryptography, Modular Multiplication, NTRU, NTT, Fully Homomorphic Encryption.*

1. INTRODUCTION

Over the last decades, information technologies (IT) have affected our everyday lives. We expect this impact to increase in the next years. The dependence on IT also increases the need to secure the enormous amounts of sensitive data exchanged or stored through public networks and data centers. Actually, many companies and organizations opt for cloud computing solutions. The cloud provides many options, from basic storage to applications and services. This choice enables high scalability, quick deployment, dynamic resources, high computing power, etc, but what about data confidentiality and user privacy in the cloud?

To overcome these problems, the providers can use the Fully Homomorphic Encryption (FHE) schemes.

The FHE is a breakthrough new technology, which allows the Cloud Server to perform efficient computation over encrypted data and provides the result computed as cipher-texts to Clients. The result of such a computation remains in encrypted form, only the Clients can decrypt this result computed by the Cloud Server.

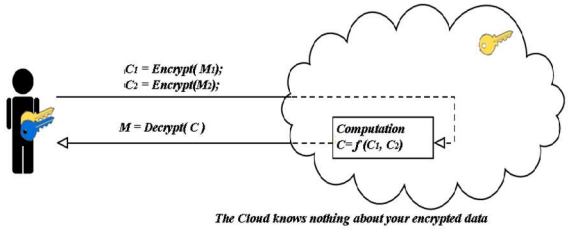

Figure.1: Computation on encrypted data by the Cloud Server.

This technique envisaged first by Rivest, Adleman, and Dertouzos 30 years ago. In 2009 Craig Gentry [1], constructed the first FHE scheme based on Ring Learning With Errors (Ring-LWE) problems. Since Gentry's proposition, many FHE schemes based on the Ring Learning with Errors RLWE) or NTRU were proposed; the major challenge was building efficient schemes in the practice.

The main basic operations over encrypted data are addition and multiplication. A scheme which is both additively and multiplicatively Homomorphic is called Fully Homomorphic Encryption (FHE), this standard definition allows more general constructions, and thus, an FHE scheme enables computation of arbitrary functions on encrypted data. Formally, if c_1 (respectively c_2)

is a cipher of m_1 (respectively m_2) there are two operations such as:

Additive Homomorphic Encryption:

$$Dec(c_1 \oplus c_2) = Dec(c_1) \oplus Dec(c_2) = m_1 + m_2. \quad (1)$$

Multiplicative Homomorphic Encryption:

$$Dec(c_1 \otimes c_2) = Dec(c_1) \otimes Dec(c_2) = m_1 * m_2. \quad (2)$$

If the scheme is only additively or multiplicatively homomorphic, we call the scheme Partially Homomorphic. For most FHE schemes, the multiplicative depth of circuits is the main practical limitation in performing computation over encrypted data.

1.1 Contributions

Firstly, we contribute by creating a new NTRUrobust_PKE (Public Key Encryption) release inspired by our last paper titled "New Efficient and Robust NTRU Post-Quantum Key Exchange release- NTRUrobust" [3] and NTRU scheme [4]. This release implements the NTT algorithm combined with our own Fast Modular Multiplication Algorithm (FMMA) for speeding-up the polynomial multiplication. Compared to the use of the convolution multiplication, our implementation grows the speed performance of the cryptographic process by factor up to 87 times for keys generation, 35 times for encryption, and 47 for decryption.

Secondly, we contribute by creating a new Fully Homomorphic Encryption (FHE) scheme over NTRUrobust_PKE. Our FHE protocol performs iteratively the addition, the multiplication, and combination of them, over the encrypted data for unlimited depths, and with perfect correctness.

The principal idea of our purpose is to build the FHE scheme by "Encryption/Computation/ Decryption/Iteration" (ECDI) for each step described in section 4.

We define the NTRUrobust_PKE in the polynomial ring of the form $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ with the parameters set {n=1024, q=65537, p=2} that achieving the category 5 security level defined by NIST "the category 5 security level is equivalent to an algorithm is at least as hard to break AES256"[5].

1.2 Outline

We organize the rest of our work as follows: the section.1 contains this introduction; in Section 2, we define some notation used in this work, we describe briefly the FMMA algorithm, and NTT algorithm; and we recall some related works to the Fully Homomorphic Encryption; the Section 3, concerns the description of our NTRUrobust public-key encryption scheme; in Section 4, we present our FHE scheme; in Section 5, we present the result analysis of NTRUrobust_FHE implementation and FHE scheme implementation ; in Section 7, we give a conclusion on our work and our future research orientation.

2. PRELIMMINARIES

In this section, we will focus to provide only descriptions of the principal subjects that we will evoke in this work. So, we describe briefly our Fast Modular Multiplication and the NTT algorithm and for more details see [1].

2.1 Notation

In the remainder of this paper, we use the following notations: $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ the polynomials ring ; ***(a,b,..)*** uppercase the elements of R_q; $(a,b,\ldots)$ lower-case the coefficients of elements of R_q; (*) denotes the multiplication of two polynomials; $(\cdot)$ denotes the multiplication of two integer; $\hat{\boldsymbol{f}} = NTT(\boldsymbol{f})$ the function that transforms a polynomial into NTT form and the $\boldsymbol{f} = invNTT(\hat{\boldsymbol{f}})$ the inverse NTT function that transforms the polynomial to normal form; we also note $\sum_{i=0}^{N-1} \hat{h}_i \; X = \sum_{i=0}^{N-1} \hat{f}_i * \hat{g}_i \pmod q$, the point wise multiplication of two polynomials in NTT form and we also note it $\hat{\boldsymbol{h}} = \hat{\boldsymbol{f}} \circ \hat{\boldsymbol{g}}$; we refer to sampCBD(***seed***) the polynomial sampled according to Centered Binomial Distribution; the convolution multiplication (*) of two polynomials $\boldsymbol{f}$ and g is:

$$\boldsymbol{f} * \boldsymbol{g} = \sum_{i+j=k}^{N} f_i \, g_j \, X^k; \text{ With } i, j, \; k \in [0, N[. \quad (3)$$

2.2 Fast Modular Multiplication Algorithm

We construct FMMA especially for Fermat prime numbers and all numbers of the form: $q = 2^k + 1$. In our case study, we use the fourth Fermat prime number as the modulus for our implementation, as we will describe below [6].

FMMA is more suitable for computing modular multiplication by transforming the modulus parameter q, into a number of the form $\phi = q - 1 = 2^k$, because, this form allows the computers, signal processors, and microprocessors to speed-up the reduction and the division by simple logical operators (&), and (>>) respectively.

Algorithm1: FMMA

Input: *Integers* x, y, *modulus* $q = 2^k + 1$, *and* $\phi = (q-1)$.

FMMA( ):

1. $p \leftarrow \quad x * y;$
2. $z \leftarrow \quad p \ .\& \ (\phi - 1);$
3. $d \leftarrow \quad (p - z) >> log_2(\phi);$
4. $result \leftarrow \quad (z - d);$
5. $if (result < 0)\ then\ return\ result + q.$
6. $else\ return\ result.$

Output: *Reduced number:* $result = x * y \pmod{q}$.

2.3 Number Theoretic Transform (NTT)

The number-theoretic transform (NTT) is a generalization of the Discrete Fourier Transform (DFT), see [7, 8], which is carried out in positive Integer group and finite fields whereas the DFT is defined in complex numbers group.

The Number Theoretic Transform (NTT) provides efficient polynomials multiplication in the form's ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ (with n power of two and q prime number). NTT has many applications in the computer arithmetic and cryptographic domain, because it reduces the time complexity from $O(n^2)\ to\ O(n * log(n))$.

To use NTT algorithm, we must choose the modulus that satisfying, q=$k.n$+1 then the multiplicative group $\mathbb{R}^n$ has size $\varphi(q) = q - 1 = k.n$, a generator g and computing the primitive nth root of unity Omega $\omega = g^k (\mathrm{mod} q)$ and $\omega^n = g^{kn} = 1(\mathrm{mod} q)$.

A polynomial in normal form:

$$\boldsymbol{f} = \sum_{i=0}^{n-1} f_i X^i \in R_q \ (mod\ q). \qquad (4)$$

Can be transformed to NTT form by:

$$\hat{\boldsymbol{f}} = \sum_{j=0}^{n-1} \gamma^j f_j \omega^{ij} \quad (mod\ q). \qquad (5)$$

With $\gamma = \sqrt{\omega}$ is the 2nd root of unity.

And the inverse of NTT function to return back to normal form is computed by:

$$f_i = n^{-1}\gamma^{-i} \sum_{j=0}^{n-1} \hat{f}_j\ \omega^{-i} \quad (\mathrm{mod} q). \qquad [6]$$

With $invNTT\ (\hat{\boldsymbol{f}}) = \boldsymbol{f}$.

In our implementation, the FMMA function is integrated in NTT functions and it is called for each coefficients multiplication. For more details of FMMA and NTT algorithms and how to implement them, the reader can see our paper [3].

2.4 Related Works

The homomorphic encryption includes different methods that perform distinct classes of computations. (1) The partially homomorphic encryption (PHE) that supports only one computation like RSA which supports only the multiplication operation on ciphertexts; (2) the Somewhat homomorphic encryption (SHE) can evaluate the addition and multiplications operations but only for limited depth; (3) the Leveled fully homomorphic encryption (LFHE), can evaluate both operations but only for limited depth (pre-determined); (4) the FHE can evaluate all operations of unlimited depth.

Many works are performed for creating FHE cryptosystems. In this section we are going to give a brief description of the homomorphic encryption for RSA scheme and for some other lattice-based schemes.

2.4.1 RSA homomorphic encryption

RSA is partially homomorphic because it supports the multiplication only; the evaluation of homomorphic encryption is not suitable for addition.

So, let be ***N, e, d*** are the RSA parameters and $\boldsymbol{c_1}$, $\boldsymbol{c_2}$ two ciphertexts respectively of $\boldsymbol{m_1}$ and $\boldsymbol{m_2}$. The multiplicative homomorphic encryption is defined as follow:

$$c = c_1 * c_2 = m_1^e * m_2^e = (m_1 * m_2)^e \ (mod\ N).$$

Then the decryption of c is:

$\mathrm{Dec}(c_1 * c_2) = (m_1 * m_2)^{ed=1} (\mathrm{mod}\ N)$.

But it is not additively homomorphic, because

$$\mathrm{Dec}(c_1 + c_2) = (m_1^e + m_2^e)^d \neq (m_1 + m_2) \ (\mathrm{mod}\ N).$$

2.4.2 Gentry FHE cryptosystem

Craig Gentry, cryptosystem [2] is defined in the ring of the form $R_q = \mathbb{Z}_q[X]/(X^N + 1)$.

Gentry chooses an error $\boldsymbol{e}$ in an ideal $\boldsymbol{I}$ in $\boldsymbol{R_q}$, with $\boldsymbol{e} = k\boldsymbol{I}$ and compute a ciphertext by: $\boldsymbol{c} = \boldsymbol{m} + k\boldsymbol{I}$.

So the encryption of two message $\boldsymbol{m_1}$ and $\boldsymbol{m_2}$ is:

$\boldsymbol{c_1} = \boldsymbol{m_1} * k_1\boldsymbol{I}\ \boldsymbol{et}\ \boldsymbol{c_2} = \boldsymbol{m_2} * k_2\boldsymbol{I}$

The addition of two ciphertext as:

$$\boldsymbol{c_1} + \boldsymbol{c_2} = \boldsymbol{m_1} + \boldsymbol{m_2} + (\boldsymbol{k_1} + \boldsymbol{k_2})\boldsymbol{I}$$

Then we can decrypt: $\boldsymbol{Dec}(\boldsymbol{c_1} + \boldsymbol{c_2}) = \boldsymbol{m_1} + \boldsymbol{m_2}$.

And the multiplication of two ciphertexts as:

$\boldsymbol{c_1} \otimes \boldsymbol{c_2} = \boldsymbol{m_1} \otimes \boldsymbol{m_2} + ((\boldsymbol{m_1} \otimes \boldsymbol{m_2})\boldsymbol{k_1} \oplus \boldsymbol{k_1}\boldsymbol{k_2})\boldsymbol{I}$.

The multiplication represents an enormous challenge to build a FHE because the error is very affected by the multiplication. So, Gentry got the FHE from the SHE by using a method called "Bootstrapping", which comprises testing the decryption function step by step [9].

2.4.3 Brakerski-Gentry-Vaikuntanathan (BGV)

The BGV cryptosystem [1] is based on Ring-LWE

problem, and they define it in the polynomials ring of the form $R_q = \mathbb{Z}_q[X]/(X^n+1)$.

The BGV begins by generating randomly a private key ***s***, an error ***e***, and ***a*** from the ring R_q; and computing $(a, b) = (a, a * s + p * e)$ where ***b*** is the public key.

For encryption, the system chooses a message M, and chooses randomly ***(s1, r1, r2)*** elements of ***Rq*** ; and computing the pair of ring elements ***(u,v)= (a *s1+r1, b*s1+p*r2+M)***.

And for decryption the system performs $m' = u * s + v \pmod q$, and computes ***m=m'*** *(mod p)* in ***Rp***.

So, it is easy to verify the BGV homomorphic encryption for addition:

c1+c2=(u1+u2,v1+v2)=(u,v) *and* ***Dec(c1+c2)=m1+m2;***

But the BGV HE for multiplication is very difficult. Let be :

c1=(u1,v1)= (a *s1+r1, *b.s1+p.r2+m1);*

c2 =(u2,v2)= (a *s2+r3, *b.s2+p.r4+m2);*

The multiplication of ***c1*** and ***c2*** is:

c= c1*c2=(u1*v1, u1*v2+u2*v1, u2*v2)=(u,v,w).

We obtain a vector with 3 elements. To tackle this, the BGV authors "use a modulus reduction technique, which uses progressively smaller modulus qℓ for each level ℓ and rescales the ciphertext to the smaller modulus to reduce its noise level" [9].

2.4.4 NTRU multi-keys FHE

The Adriana L´opez-Alt et al. [10] multi-key FHE scheme allows the computation over encrypted data with multiple keys and multiple users.

The multi-key FHE scheme is based on NTRU scheme and defined in the form's ring $R_q = \mathbb{Z}_q[X]/(X^N+1)$. The system begins by generating two elements ***f***, ***g*** and computes the private key ***F=2f+1***; computing the inverse of ***F*** in ***Rq***; computing the public key ***h= 2g/F*** *mod q*. For the encryption of a message ***m,*** the system generates ***r***, ***e***, and computes ***c=h*r+2e+m*** *mod q*. And for decryption of ***c,*** the system computes ***a = F*c*** *mod q*, and ***m=a*** *(mod 2).*

The FHE principle is to compute a keys (***hi, Fi***) for each message ***mi***. So the Additive homomorphic encryption can be easily verified: ***cadd*** = ***c1 + c2*** *mod q* should be decrypted to ***m1 + m2*** by ***F1*** and ***F2***. But the multiplicative homomorphic encryption is very difficult: ***cmult = c1 *c2*** *mod q* should be decrypted to ***m1* m2*** by ***F1*** and ***F2*** . So:

Dec(cmult, F1,F2)= F1*F2 (c1*c2)

= 2[2g1*g2*r1*r2 + g1*r1*F2(2e2 + m2) + g2*r2*F1(2e1 + m1)+F1*F2 (e1*m2 + e2*m1 + 2e1*e2)] + F1*F2 (m1*m2)

= 2emult+F1*F2 (m1*m2) . (6)

The correctness decryption of ***cadd*** and ***cmult*** turn on "the sum and the product of the underlying messages, respectively, as long as the error (***emult***) does not grow too large"[10].

3. OUR NTRU_ROBUST_PKE

In 1996, NTRU was introduced by the three mathematicians J. Hofstein, J. Pipher, and J. H. Silverman, and then published in 1998 [11]. They presented it as an alternative to RSA and ECC. NTRU is completely constructed from Lattice-Based-Cryptography [12]. Its domain of computation is the polynomials ring of the form $R_q = \mathbb{Z}_q[X]/(X^n-1)$ with n is a prime number and the modulus q is a power of two, or the polynomials ring of the form $R_q = \mathbb{Z}_q[X]/(X^n+1)$ with n is a power of two and q is a prime number. Since its first creation there are several versions, the latest NTRUhps scheme is now a candidate to NIST's post-quantum standardization project. According to the NIST experts' analysis, NTRU is an exciting field of research, and it is very efficient [13] (for more details see [4]).

In this section we describe our NTRUrobust_PKE public key encryption release inspired from our latest NTRUrobust post-quantum key exchange release [3].

3.1 NTRUrobust_PKE parameters definition

Our NTRUrobust_PKE scheme implements the NTT algorithm combined with our FMMA algorithm for speeding-up the polynomials multiplication. We define it in the polynomials ring of the form $R_q = \mathbb{Z}_q[X]/(X^N+1)$, with the parameters set that achieves the category 5 security level defined by NIST $\{n = 1024,\ q = 65537,\ p = 2,\ \sigma = \sqrt{16/2}\}$, with σ is the standard deviation used for choosing the polynomials according to Centered Binomial Distribution. The modulus used is the fourth Fermat prime number $q = 2^{2^4}+1 = 65537$. The Fermat prime numbers were first studied by Pierre Fermat [12].

To use the NTT algorithm we must defining the n-th primitive root of unity omega: $\omega = (\text{mod } q) = 1089$, its square root gamma: $\gamma = \sqrt{\omega} = 33$, the inverse of gamma: $\gamma^{-1} \pmod q = 1986$, the inverse of

omega: $\omega^{-1} \pmod{q} = 11976$, and the inverse of n modulo q is $n^{-1} \pmod{q} = 65473$.

The polynomials are sampled according to Centered Binomial Distribution (sampCBD() function) Alkim et al. [14], in the sets {*f, g, r, m*} defined respectively as below:

$$\boldsymbol{L_f} = \{\boldsymbol{f} \in \boldsymbol{R}\ with\ f_i \in [-3,..,3]\};$$
$$\boldsymbol{L_g} = \{\boldsymbol{g} \in \boldsymbol{R}\ with\ g_i \in [-3,..,3]\};$$
$$\boldsymbol{L_r} = \{\boldsymbol{r} \in \boldsymbol{R}\ with\ r_i \in [-3,..,3];$$
$$\boldsymbol{L_m} = \{\boldsymbol{m} \in \boldsymbol{R}\ with\ m_i \in \{0,1\}\}.$$

We also note that our version takes the private key F in the form $\boldsymbol{F} = p.\boldsymbol{f} + 1$ [15]. This form allows us to avoid the computation of the inverse of f *(mod p)* because $\boldsymbol{F} = p.\boldsymbol{f} + 1 \pmod{p} = 1$

The Keys Generation, Encryption, and Decryption algorithms below illustrate the use of the NTT functions, and we note that we integrate the FMMA modular multiplication function into NTT functions for each polynomial coefficients multiplication [3][7].

3.2 Algorithm 2: Keys Generation

Input: the Integer parameters $n, q, p, and\ seed$.

1. $\boldsymbol{f}, \boldsymbol{g} \leftarrow sampCBD(\boldsymbol{seed})$;
2. $\boldsymbol{F} \leftarrow \boldsymbol{F} = p.\boldsymbol{f} + \boldsymbol{1}$;
3. $\widehat{\boldsymbol{F}}, \widehat{\boldsymbol{g}} \leftarrow \boldsymbol{NTTfunc}(\boldsymbol{F}, \boldsymbol{g})$;
4. $inv\widehat{\boldsymbol{F}} \leftarrow \dfrac{\boldsymbol{1}}{\boldsymbol{F}} \pmod{q}$;
5. $\widehat{\boldsymbol{h}} \leftarrow \widehat{\boldsymbol{g}} \circ inv\widehat{\boldsymbol{F}} \pmod{q}$;

Output: private key $\widehat{\boldsymbol{F}}$ and the public key $\widehat{\boldsymbol{h}}$ saved in NTT form**.**

Comment: In the key generation our implementation generates both private keys (f, g) at the same time by the sampCBD(seed) function(line 1) that allows us to increase the key generation process, this function uses the SHAKE-256 Keccak hash function [16]. The implementation keeps the private key and the public key in NTT form. In (line 4), the inverse of the polynomial ($\widehat{\boldsymbol{F}}$) is found by computing the inverse of each coefficient modulo $\boldsymbol{q}$, $inv\widehat{F}_i = 1/\widehat{f}_i \pmod{q}$ by using the extended Euclidean algorithm.

3.3 Algorithm 3 : Encryption

Input : The public key $\widehat{\boldsymbol{h}}$, message ***msg,*** and ***seed***

1. $\boldsymbol{m} \leftarrow Map(\boldsymbol{msg})$;
2. $\widehat{\boldsymbol{m}} \leftarrow NTTfunc(\boldsymbol{m})$;
3. $\boldsymbol{r} \leftarrow sampCBD(\boldsymbol{seed})$;
4. $\widehat{\boldsymbol{r}} \leftarrow NTTfunc(\boldsymbol{r})$;
5. $\widehat{\boldsymbol{c}} \leftarrow p.(\widehat{\boldsymbol{r}} \circ \widehat{\boldsymbol{h}}) + \widehat{\boldsymbol{m}}$;

Output: The encrypted message $\widehat{\boldsymbol{c}}$ in NTT form.

Comment: In (line 1), the encryption function maps the message into a binary polynomial. In (line.5) the cipher-text is computed by using point-wise Multiplication (○) of (r̂ and ĥ), multiplying them by the parameter p, and adding the message m̂. All polynomials computed are now in the NTT form.

3.4 Algorithm 4: Decryption

Input: The encrypted message $\widehat{\boldsymbol{c}}$**,** and the private key $\widehat{\boldsymbol{F}}$**.**

1. $\widehat{\boldsymbol{a}} \leftarrow \widehat{\boldsymbol{c}} \circ \widehat{\boldsymbol{F}} \pmod{q}$;
2. $\boldsymbol{a} \leftarrow InvNTT(\widehat{\boldsymbol{a}}) \pmod{q}$;
3. $\boldsymbol{a} \leftarrow lefting \quad a_i \in \{\dfrac{-q}{2}, \dfrac{q}{2}\}$;
4. $\boldsymbol{m} \leftarrow \boldsymbol{a} \pmod{p}$;

Output: The message ***m*** decrypted in binary polynomial in the normal form**.**

Comment: In (line1), the decryption function computes a polynomial $\widehat{\boldsymbol{a}}$ in NTT form by the point-wise multiplication of the ciphertext $\widehat{\boldsymbol{c}}$ and the private key $\widehat{F}$, in (lin 4) the first step is achieved by computing the decrypted message ***m*** in normal form.

The NTRUrobust_PKE warrants perfect correctness of the decryption function with failure probability rate equal to ZERO ($2^{-\infty}$), even we generate the private key in the form $\boldsymbol{F} = p.\boldsymbol{f} + 1$ [15], The result is obtained by using the python script developed by NTRU team [17, 18]. So NTRUrobust_PKE is very confident against an eventual attack using decryption failure [19].

4. OUR FULLY HOMOMORPHIC ENCRYPTION SCHEME (FHE)

We construct the FHE scheme over our NTRUrobust_PKE, which computes the private key, the public key, the ciphertext, and the plaintext in the NTT form by using point-wise multiplication. So the sum and the product of two

Journal of Theoretical and Applied Information Technology
15th March 2021. Vol.99. No 5

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

encrypted data performed by our FHE scheme, not increase largely the error which allows the decryption to be perfect and more exact than the NTRU scheme that uses the convolution multiplication. Let be:

$$\hat{c}_1 = p.(\hat{r}_1 \circ \hat{h}) + \hat{m}_1 \ (mod\ q). \quad (7)$$
$$\hat{c}_2 = p.(\hat{r}_2 \circ \hat{h}) + \hat{m}_2 \ (mod\ q).$$

- The homomorphic encryption for addition of two encrypted data is as follow:

$$\hat{c} = \hat{c}_1 + \hat{c}_2 = p.\left(\hat{h} \circ (\hat{r}_1 + \hat{r}_2)\right) + \hat{m}_2 + \hat{m}_1 (\text{mod q}). \quad (8)$$

Then we calculate a ciphertext coefficient $\hat{c}_i$ of the polynomial $\hat{c}$ by:

$$\hat{c}_i = \hat{c}_{1,i} + \hat{c}_{2,i} = p.\hat{h}_i.(\hat{r}_{1,i} + \hat{r}_{2,i}) + \hat{m}_{1,i} + \hat{m}_{2,i}.$$

With $\hat{c}_{1,i}, \hat{c}_{2,i}, \hat{r}_{1,i}, \hat{r}_{2,i}, \hat{m}_{1,i}$, and $\hat{m}_{2,i}$ are the coefficients respectively of the polynomials $\hat{c}_1, \hat{c}_2, \hat{r}_1, \hat{r}_2, \hat{m}_1$, and $\hat{m}_2$.

Our implementation computes easily the decryption of the addition of two encrypted data by computing a polynomial $\hat{a} = \hat{c} \circ \hat{F}\ mod\ q$; computing $a = invNTT(\hat{a})$; and computing $m = a\ mod\ p$.

Then: $Dec(c = c_1 + c_2) = m = m_1 + m_2$.

- The homomorphic encryption for Multiplication of two encrypted data is as follow:

$$\hat{c} = \hat{c}_1 \circ \hat{c}_2 = (p.(\hat{h} \circ \hat{r}_1) + \hat{m}_1) \circ (p.(\hat{h} \circ \hat{r}_2) + \hat{m}_2) (\text{mod q}). \quad (9)$$

We calculate a ciphertext coefficient $\hat{c}_i$ of the polynomial $\hat{c}$ by:

$$\hat{c}_i = \hat{c}_{1,i} * \hat{c}_{2,i} = (p.\hat{h}_i.\hat{r}_{1,i} + \hat{m}_{1,i})(p.\hat{h}_i.\hat{r}_{2,i} + \hat{m}_{2,i})$$
$$= p.\hat{h}_i(p.\hat{h}_i\hat{r}_{1,i}\hat{r}_{2,i} + \hat{r}_{1,i}\hat{m}_{2,i} + \hat{r}_{2,i}\hat{m}_{1,i}) + \hat{m}_{1,i}\hat{m}_{2,i}$$

Our implementation computes easily the decryption of the product of two encrypted data by computing a polynomial $\hat{a} = \hat{c} \circ \hat{F}\ mod\ q$, computing $a = invNTT(\hat{a})$, and computing $m = a\ mod\ p$.

Then: $Dec(c = c_1 * c_2) = m = m_1 * m_2$.

The NTRUrobust_PKE decrypt exactly the computation of two encrypted data. So, for computing (addition, multiplication, and combination of them) many encrypted data, our FHE scheme apply the process **E**ncrypt/ **C**ompute /**D**ecrypt/**R**e-**E**ncrypt/**I**teration (ECDREI) for each depth.

For example, if we want to encrypt m_1,m_2, and m_3 in c_1, c_2, c_3 and we want the Cloud Server to compute $c_1*c_2*c_3$; the process is : the Client sends (c_1, c_2) to the Cloud Server; the Cloud Server computes $c=c_1*c_2$ and sends it to Client; the Client decrypts **c** in $m=m_1*m_2$ and **re-encrypt** it in c; the next step the Client send c and c_3 and the Cloud Server computes $c= c*c3$ and return it to Client who decrypts it in $m=m*m_3= m_1*m_2*m_3$.

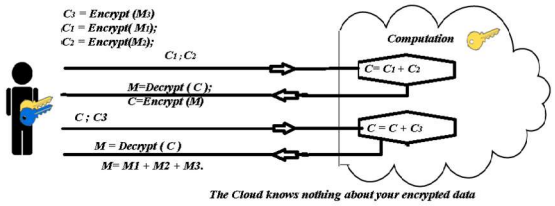

Figure.2: Iterative computation of three encrypted data.

In the subsections below we will present the Additive Homomorphic Encryption (AdditiveHE) algorithm, the Multiplicative Homomorphic Encryption (MultiplicativeHE) algorithm, and the combination of both Additive and Multiplicative Homomorphic Encryption (AddMulFHE) Algorithm that computes:
$\text{Dec}(\mathbf{a_1} * \mathbf{b_1} + \mathbf{c_1} * \mathbf{d_1}) = \mathbf{ma_1} * \mathbf{mb_1} + \mathbf{mc_1} * \mathbf{md_1}$.

With $\mathbf{a_1}, \mathbf{b_1}, \mathbf{c_1}$, and $\mathbf{d_1}$, the encrypted data respectively of the messages $\mathbf{ma_1}, \mathbf{mb_1}, \mathbf{mc_1}$, **and** $\mathbf{md_1}$.

4.1 Additive Homomorphic Encryption

In algorithm.5, we suppose that a ClientA send the encrypted data c_i of the messages m_i, and the Cloud Server computes $\sum_{i=1}^{i=depth} c_i$; and then the ClientA obtains the result of the sum $\sum_{i=1}^{i=depth} m_i$. NTRUrobust_PKE cryptosystem performs the AdditiveHE, by computing **iteratively** the addition of many encrypted data, and following the process **E**ncrypt/ **C**ompute /**D**ecrypt/**R**e-**E**ncrypt/**I**teration (ECDREI) for each depth.

$$Decrypt\left(\sum_{i=1}^{i=depth} c_i\right) = \sum_{i=1}^{i=depth} m_i\ . \quad (10)$$

Algorithm 5: AdditiveHE Algorithm.

Input: The public key $\hat{h}_a$, the private key $\hat{F}_a$, the mapped messages $m_i \in R_2$ *(binary polynomial)*, and ***depth.***

```
1  m ← m1;
2.  for(int i = 1; i ≤ depth; i++) do : {
3.      m̂ ← NTT(m);
4.      m̂i ← NTT(mi);
5.      ĉa ← encryption(ĥa, m̂);
6.      ĉi ← encryption(ĥa, m̂i);
7.      ĉa ← Add(ĉa, ĉi); //Server
```

8. $m \leftarrow$ ***decryption***$(\hat{c}_a, \hat{F}_a)$;
9. } *end for*;
10. return m;

Output: The Decrypted message $m = \sum_{i=1}^{dept} m_i \bmod p$ in normal form.

Comment: So, in line.1 the AdditiveHE initializes the binary polynomial $m=m_1$ which will stores the result; in line.2 the first **iteration** begins the addition polynomial (messages); ClientA transforms (m, m_2) to NTT form (lines 3,4), encrypts them in $(\hat{c}_a, \hat{c}_2)$ respectively (lines 5,6), and sends them to the Server; the Server computes $\hat{c}_a = \hat{c}_a + \hat{c}_2$ (line.7) and sends it to ClientA; the ClientA decrypt $\hat{c}_a$ in **m** which equal to $m=m_1+m_2$; and so on until ***depth*** limit.

4.2 Multiplicative Homomorphic Encryption

In algorithm.6, we suppose that the ClientA sends the encrypted data c_i of the messages m_i to the Cloud Server, and the Cloud Server computes $\prod_{i=1}^{depth} c_i$; and then the ClientA obtains the result of the product of $\prod_{i=1}^{depth} m_i$. NTRUrobust_PKE cryptosystem performs the multiplicative homomorphic encryption called (MultiplicativeHE), by computing **iteratively** the multiplication of many encrypted data by following the process **E**ncrypt/ **C**ompute /**D**ecrypt/**R**e-**E**ncrypt/**I**teration (ECDREI) for each depth.

$$Decrypt\left(\prod_{i=1}^{dept} c_i\right) = \prod_{i=1}^{dept} m_i \,. \qquad (11)$$

Algorithm 6: MultiplicativeHE Algorithm

Input: The public key h, the mapped messages $m_i \in R_2$ *(binary polynomial)*, and ***depth.***

1. $m \leftarrow m_1$;
2. $for(int\ i = 1; i \le depth; i++)\ do : \{$
3. $\hat{m} \leftarrow NTT(m)$;
4. $\hat{m}_i \leftarrow NTT(m_i)$;
5. $\hat{c}_a \leftarrow encryption(\hat{h}_a, \hat{m})$;
6. $\hat{c}_i \leftarrow encryption(\hat{h}_a, \hat{m}_i)$;
7. $\hat{c}_a \leftarrow \hat{c}_a \circ \hat{c}_i$;//Server
8. $m \leftarrow \boldsymbol{decryption}(\hat{c}_a, \hat{F}_a)$;
9. } *end for*;
10. *return* m;

Output : The Decrypted message $m = \prod_{i=1}^{k} m_i \bmod p$ in Normal form.

Comment: In line.1 the MultiplicativeHE initializes the binary polynomial $m=m_1$ which will store the result; in line.2 the first **iteration** begins ; ClientA transforms (m, m_2) to NTT form $(\hat{m}, \hat{m}_2)$ (lines 3,4), encrypts them in $(\hat{c}_a, \hat{c}_2)$ respectively(lines 5,6), and sends them to the Server; the Server computes $\hat{c}_a = \hat{c}_a \circ \hat{c}_2$ (line.7) by point-wise multiplication because they are in NTT form; and sends $\hat{c}_a$ to ClientA; the ClientA decrypt $\hat{c}_a$ in **m** in normal form, which equal to $m=m*m_2$; and so on until ***depth*** value.

4.3 Combination of Additive and Multiplicative Homomorphic Encryption

We can combine the AdditiveHE algorithm and MultiplicativeHE algorithm for computing many equations that we want. For example, we can use those algorithms for computing the equation below:

$\text{Dec}(ca * cb + cc * cd) = (a * b + c * d)$.

Where (ca, cb, cc, cd) are the encrypted data (polynomials) of (a, b, c, d) respectively.

Algorithm 7 : CombineFHE Algorithm

Input: The public key $\hat{h}_a$, the private key $\hat{F}_a$, and the mapped messages (a, b, c, d) *in* R_2^4.

1. $\hat{a}, \hat{b}, \hat{c}, \hat{d} \leftarrow \text{NTT}(a, b, c, d)$; //ClientA
2. $\widehat{ca}, \widehat{cb}, \widehat{cc}, \widehat{cd} \leftarrow \text{encrypt}(\hat{h}_a, \hat{a}, \hat{b}, \hat{c}, \hat{d})$;
3. $\widehat{Cab} \leftarrow \text{MultipicativeHE}(\widehat{ca}, \widehat{cb})$; //server
4. $\widehat{Ccd} \leftarrow MultiplicativeHE(\widehat{cc}, \widehat{cd})$;//server
5. $\widehat{CA} \leftarrow \text{AdditiveHE}(\widehat{Cab}, \widehat{Ccd})$;//server
6. **Ma** $\leftarrow decryption(\widehat{CA}, \widehat{Fa})$; // ClientA

Output: The Decrypted message $Ma = (a * b + c * d)$.

Comment: In line.1 the ClientA transforms the polynomials $a, b, c,$ and d from normal form to NTT form respectively into $\hat{a}, \hat{b}, \hat{c}$, and $\hat{d}$; encrypts them (line.2) respectively into $\widehat{ca}, \widehat{cb}, \widehat{cc}$, and $\widehat{cd}$; and sends them to the Cloud Server, who in line.3 and line.4 calls the MultiplicativeHE algorithm for computing $\widehat{ca} \circ \widehat{cb}$ and $\widehat{cc} \circ \widehat{cd}$ respectively into $\widehat{Cab}$ *and* $\widehat{Ccd}$; in line.5 the Cloud Server calls the AdditiveHE algorithm for computing the sum of $\widehat{Cab}$ *and* $\widehat{Ccd}$ into $\widehat{CA}$ obtained in NTT form and return it to ClientA; finally, ClientA decrypts the encrypted message $\widehat{CA}$ into **Ma**, which is computed in normal form, and represent the final result **Ma** $= (a * b + c * d)$.
If we have to compute the sum or the product of **Ma** for many times, we should call iteratively this algorithm until the pre-defined ***depth***.

5. RESULT ANALYSIS

We note that all implementations are performed on a PC-TOSHIBA with an Intel(R) Core(TM) i7-2630QM CPU, 2 GHz processor, RAM 8 GO, under environment Windows 7-32 bits and Dev-C++ 4.9.9.2; and the speed of the algorithms are given in milliseconds (ms).

5.1 Performance of our NTRUrobust_PKE

In this subsection we present the performance results of our *NTRUrobust_PKE* using (NTT&FMMA) compared to the same NTRU scheme but implemented with **convolution multiplication**, we called it **NTRU_CONV.**

Both implementations use the same parameters set *{n=1024, q=65537, p=2}*. We choose this sequence parameters for targeting the high security level that meets the category 5 defined by NIST, and the Fermat prime number $q=2^16+1=65537$, allows the modular multiplication to be fast by using our FMMA algorithm.

We built the two software on the same machine cited above, and we report the median performance results of 100 runs in Table 1(Figure.3).

Table 1: Speed performance benchmarking (ms)

Schemes	KeysGen (ms)	Encrypt (ms)	Decrypt (ms)
NTRU _CONV	109	16	15
NTRUrobust NTT+FMMA	1.25	0.46	0.32
Speed-up Factor (X)	87 times	35 times	47 times

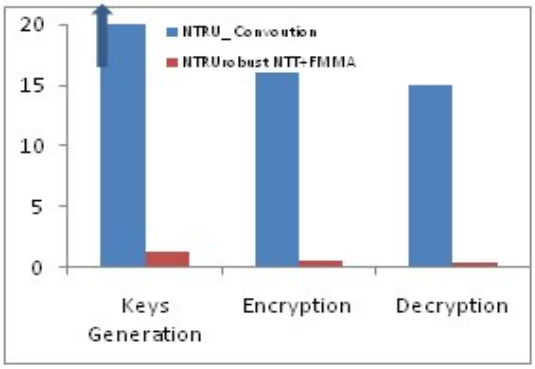

Figure 3: Performance benchmarking between NTRUrobust and NTRU_CONV. The result values are given in milliseconds (ms).

Comment: In this result, we remark that we did better, by speeding-up the key generation by a factor up to 87 times, the encryption by a factor up to 35 times, and the decryption by a factor up to 47 times.

The implementation of NTRUrobust_PKE release described in this paper is available on the Google Drive website at [20]; and the NTRU, implementations are available on the NIST website at [21].

5.2 Performance of our FHE scheme

In this subsection, we present the performance of our FHE scheme, by performing the AdditiveHE algorithm and the MultiplicativeHE algorithm for some value of depth. For example, we choose the depth value {100, 200, 400, 800, 1000}; we test both algorithms by those values, that allows us to be confident about the speed performance and the correctness of the decryption after many computation of encrypted data.

We reported the result in Milliseconds (ms), as indicated in Table.2, and Figure.4 below.

Table 2: Speed performance of our FHE algorithms (ms)

depth	**100**	**200**	**400**	**800**	**1000**
MultiplicativeHE	**124**	**256**	**516**	**1030**	**1201**
AdditiveHE	**109**	**234**	**453**	**904**	**1138**

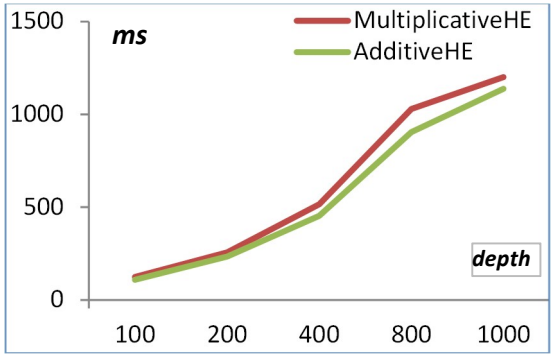

Figure 4: Complexity time(ms) evolution of the AdditiveHE and MultiplicativeHE in function of depth value. The result values are given in milliseconds.

Comment: We did those tests with large depths to prove the efficiency of AdditiveHE and MultiplicativeHE algorithms of our FHE scheme based on the process Encrypt/ Compute /Decrypt/Re-Encrypt/Iteration (ECDREI). Thanks to our NTRUrobst_PKE cryptosystem, we obtain the perfect correctness of the decryption function

for all tests, which gives us great confidence in our implementation.

We remark, in figure.4, that the evolution is in linear time with depth's values. The complexity of the AdditiveHE and MultiplicativeHE algorithms are almost the same, which proves the efficiency of our Fully Homomorphic Encryption scheme. Whereas Gentry's homomorphic cryptosystem is not applicable given the increased noise for addition and the multiplication; DGHV, Homomorphic cryptosystem, the encryption-decryption time remains very large; and for NTRU-multi-Keys FHE will be very complicated when the depth value is large. For example, with the depth=100, we should create 100 private keys and 100 public keys.

Our FHE scheme over our NTRUrobust_PKE is very efficient; we create only one private key and one public key for unbounded ***depth.***

5.3 NTRUrobust_PKE Security

Many cryptanalysis works are performed; their principal goal was to check the robustness of the Lattices-Based Cryptography by posing the hardest problems on point lattice in $\mathbb{R}^n$. The best tools used to prove the security is Lattice reduction by the algorithms (Gram-Schmidt, LLL, BKZ algorithms) and Meet-in-The-Middle attack (MIM)[4] [12].

We used Martin R. Albrecht et al. [22] tool to estimate the security level of NTRUrobust_PKE release, by solving the uSVP (primal attack) with BKZ cost model of size $c=2^{0,292b}$ for the classical security and decreasing this size to $c=2^{0,265b}$ for quantum security (***b*** is the block size used by the BKZ algorithm). This tool used the quantum sieving algorithm to consider potential Grover speed-ups [14]. Our release achieves 2^{230} for the classical security level and 2^{208} for the quantum security level.

NIST states its report [8240] that "the security of NTRU is based on stronger assumption than LWE or RLWE schemes also based on Lattices" [13].

6. CONCLUSION

In this paper, we purpose complementary works; we contribute by creating NTRUrobust_PKE post-quantum (Public Key Encryption), with an excellent performance in terms of speed as well as in terms of security (the parameters set meets Category 5 security level defined by NIST). And then we create over it an efficient FHE scheme that performs the principle operations, the Additive Homomorphic encryption and the Multiplicative Homomorphic encryption, with a perfect correctness of decryption function and for unbounded ***depth***.

Thanks to our Fast Modular Multiplication Algorithm (FMMA) with the NTT algorithm, we obtain this drastic result. Our FHE scheme is very helpful to Cloud Services providers, Industrials, and companies that exchange sensitive data.

The inconvenient of our FHE scheme is that the communication time between the Client and the Cloud Server increase linearly for each level of $depth$, because the Client should decrypts and re-encrypt the result of the computation (Addition or Multiplication of encrypted data) for each level of $depth$.

For our future works, we hope to implement our NTRUrobust_PKE and our FHE scheme to the banking systems and adapting our scheme for other industrial domains like the health domain.

REFERENCES:

[1] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Jeffrey Hoffstein, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, Vinod Vaikuntanathan, "Homomorphic Encryption Standard". NIST USA 2018.

[2] Craig Gentry, "A FULLY HOMOMORPHIC ENCRYPTION SCHEME", STANFORD UNIVERSITY, USA 2009.

[3] Laaji El Hassane, Azizi Abdelmalek, "New Efficient and Robust NTRU post-quantum key Exchange-NTRUrobust-", Mohammed First University, Morocco 2020.

[4] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, "Algorithm Specifications And Supporting Documentation", Wilmington USA, 2019.

[5] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, "NISTIR 8309- Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process," NIST, USA 2020.

[6] P. Ribenboim. "The New Book of Prime Number Records", New York, Springer-Verlag, 1996.

[7] Longa, P. and Naehrig, M, "Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography", Microsoft Research USA 2019.

[8] Nayuki Project, "Number-Theoric-Transform (Integer DFT)". Link: https://www.nayuki.io/page/number-theoretic-transform-integer-dft

[9] Z.Xiaojun, X.Chunxiang, j.Chunhua, X.Run, Z.Jining, "Efficient fully homomorphic encryption from RLWE with an extension to a threshold encryption scheme", School of Computer Science and Engineering, University of Electronic Science and Technology of China 2014.

[10] Adriana L´opez-Alt, Eran Tromer, Vinod Vaikuntanathan, "On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption", New York University USA 2013.

[11] Jeffrey Hoffstein, J. Pipher, and J. H. Silverman, "Introduction Mathematics and Cryptography NTRU" Wilmington USA 1998.

[12]. Jill Pipher, J. Hofstein, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang, "Choosing Parameters for NTRUEncrypt" , Wilmington USA 2016.

[13] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone,. Status Report NISTIR 8240 on the First Round of the NIST Post-Quantum Cryptography Standardization Process. NIST, USA 2019.

[14] Alkim, E. Ducas, Poppelman, T. and Schwabe, P. "Post-quantum key exchange-NewHope", Department of Mathematics, Ege University, USA, 2019.

[15] Mohan Rao Mamdikar, Vinay Kumar and D. Ghosh, " Enhancement of NTRU public key" National Institute of Technology, Durgapur 2013.

[16] G.Assche, G.Bertoni, J.Daemen, P.Peters, and R.Van, "Keccak Hash algorithm. Radboud University,Nederlands 2016.

[17] J.Scham, NTRU team "Decryption failure script", link: https://github.com/jschanck/ntru-ephem-dfr. USA 2019.

[18] N.Howgrave Graham, N.Phong , D.Pointcheval, J.Proos, Jill Silverman, A.Singer, William Whyte, "The Impact of Decryption Failures on the Security of NTRU Encryption", NTRU Cryptosystems". Burlington, CNRS France, University of Waterloo, Canada 2019.

[19] Daniel J. Bernstein, "Comparing proofs of security for lattice-based encryption", Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607 - 7045, USA2, Horst Gortz Institute for IT Security, Ruhr University Bochum, Germany djb.at.cr.yp.to, 2019.

[20] Laaji El Hassane, Azizi Abdelmalek, Taoufi Serraj, "Link Google drive of NTRUrobust implementation"
https://drive.google.com/file/d/1Jg4C9gjAbUJxci8mLjkS6DKcJcFdk77U/view?usp=sharing
Mohammed first University Morocco 2020.

[21] NIST link website of NTRU post-quantum cryptosystem implementation. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions

[22] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer, "Estimate all the {LWE, NTRU} schemes", In Security and Cryptography for Networks - 11th International Conference, SCN 2018, volume 11035 of Lecture Notes in Computer Science, pages 351-367. Springer, 2018.