Virtually Free Randomisations of NTT in RLWE Cryptosystem to Counteract Side Channel Attack Based on Belief Propagation

Christophe Negre[1] and Mbaye Ngom[1,2]
[1]*DALI-LIRMM, Perpignan, France*
[2]*eShard, Pessac, France*

Keywords: Side Channel Attack, Belief Propagation, Number Theoretic Transform, Randomisation, Post-Quantum Cryptosystems.

Abstract: At CHES 2017, Primas, Pessl and Mangard presented an attack on RLWE cryptosystem based on Belief Propagation. The attack applies on the Number Theoretic Transform (NTT) used to decipher a message. It gathers power consumption leakage of the multiplication by roots of unity in the NTT and then applies Belief Propagation to circulate the information of all leakage nodes, until the combined leakage reveal most of the output coefficients of the NTT. In this paper we present some randomisations which either induce in NTT some random mask on values or randomly rearrange the sequence of operations. We evaluate the level of randomisation provided by the proposed counter-measures and also the effect on the processed values in the NTT. We apply Belief Propagation on the proposed randomised NTT and we study how these randomisations affect the attack. Finally we point out that a set of three combined strategies provide a high level of randomisation and a good protection against Belief Propagation attack of Primas *et al.*

1 INTRODUCTION

The progress of quantum computers combined with Shor's algorithm (Shor, 1999) is threatening cryptosystems like RSA, ECC. NIST launched a competition in 2017[1] for selecting a set of post-quantum cryptosystems. The competition is now finished, another round of competition was launch for additional digital signature scheme. There were many candidates based on Ring/Module Learning With Error (RLWE/MLWE) problem for key establishment and digital signature and some were selected. The theoretical security of these post-quantum cryptosystems were intensively analysed during these past years. Security and protection against side channel analysis might be pursued.

One component in RLWE computation is the Number Theoretic Transform (NTT) which evaluates a polynomial at the roots of unity. This is used for efficient multiplication in the ring $\mathbb{Z}_q[X]/(X^n+1)$ of RLWE cryptosystems. In 2017 Primas, Pessl and Mangard (Primas et al., 2017) proposed an attack on NTT based on Belief Propagation (Pearl, 1982). Their attack used leakage in power consumption or electromagnetic emanation in the computation of the NTT. They used Belief Propagation to gather scattered information due to leakage in the multiplication by the roots of unity, in order to recover most of the output coefficients of the NTT. This attack only requires a single trace and is an important threat on RLWE cryptosystem on embedded devices. The purpose of this paper is to propose and study counter-measures preventing this attack.

Contributions. We study a set of randomisations of NTT computation to counteract Primas *et al.*'s attack. We focus on virtually free randomisation, which means that the studied randomisation does not imply additional operations in the ring $\mathbb{Z}_q[X]/(X^n+1)$. We study the following approaches: shuffling of operations at each stage of the NTT, randomisation of the roots of unity, randomisation by random multiplicative masks, randomisation of the reduction modulo q, and random choice of the butterfly formula in NTT. For all these considered randomisations we study the effect on the data processed in NTT and provide the level of randomisation. We also simulate the Belief Propagation attack on randomised NTT to evaluate the effect of the randomisation on this attack.

[1]https://csrc.nist.gov/projects/post-quantum-cryptography

Negre, C. and Ngom, M.
Virtually Free Randomisations of NTT in RLWE Cryptosystem to Counteract Side Channel Attack Based on Belief Propagation.
DOI: 10.5220/0012857500003767
In *Proceedings of the 21st International Conference on Security and Cryptography (SECRYPT 2024)*, pages 866-875
ISBN: 978-989-758-709-2; ISSN: 2184-7711

Organisation of the Paper. In Section 2 we review some background on RLWE cryptosystem and algorithms used for NTT. In Section 3 we review Belief Propagation and the attack of Primas *et al.* (Primas et al., 2017). In Section 4 we present a set of randomisation of NTT to counteract Primas *et al.*'s attack, we analyse the impact of the randomisation on the processed value in NTT. In Section 5 we provide some simulation results of Belief Propagation attack applied on our proposed randomised NTT. Finally, in Section 6, we give some concluding remarks.

2 RLWE CRYPTOSYSTEM AND NUMBER THEORETIC TRANSFORM (NTT)

In this section we review a cryptosystem presented in (Lyubashevsky et al., 2010) based on the Ring Learning with Error (RLWE) problem. This cryptosystem will be our reference encryption scheme to analyse the proposed counter-measure on this type of cryptosystem. We also review the algorithm used for the NTT computation involved in the ring multiplication of the ciphering/deciphering operations.

2.1 RLWE Cryptosystem

In (Lyubashevsky et al., 2010) Lyubashevsky *et al.* present a public-key encryption scheme based on RLWE. Let us first recall the RLWE problem. We consider a ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$ with q a prime integer, the coefficients of an element in $\mathcal{R}_q$ are taken in $\{-\lfloor q/2 \rfloor, \ldots, \lfloor q/2 \rfloor\}$. We also consider two random distributions on $\mathcal{R}_q$:

- $a \xleftarrow{\mathcal{U}} \mathcal{R}_q$ the uniform distribution.
- $e \xleftarrow{\chi} \mathcal{R}_q$ a distribution centered at 0 with small coefficients, e.g., a discrete Gaussian distribution with a small standard deviation.

Then, the RLWE can be formulated as follows.

Definition 1 (RLWE problem.)**.** We randomly set $\mathbf{a} \xleftarrow{\mathcal{U}} \mathcal{R}_q$ and $\mathbf{s}, \mathbf{e} \xleftarrow{\chi} \mathcal{R}_q$. The RLWE problem consists in computing $\mathbf{s}$ from:

$$\mathbf{a} \text{ and } \mathbf{b} = \mathbf{a} \times \mathbf{s} + \mathbf{e}$$

It was shown in (Regev, 2009) that the LWE problem is as hard as finding short vector/basis in a lattice (the RLWE is also assumed to be difficult but to the best of our knowledge its hardness is unknown).

The public-key encryption scheme of (Lyubashevsky et al., 2010) based on RLWE problem works as follows: the plaintext is a bit string $\mathbf{m}$ of n bits $m_0, \ldots, m_{n-1}$, and it is transformed into a polynomial $\overline{\mathbf{m}} \in \mathcal{R}_q$ with coefficients $\overline{m}_i = (-1)^{m_i} \times \lfloor q/4 \rfloor$.

- **Key Generation:**
  - Private : $\mathbf{r}_1, \mathbf{r}_2 \xleftarrow{\chi} \mathcal{R}_q$
  - Public : $\mathbf{a} \xleftarrow{\mathcal{U}} \mathcal{R}_q$ and $\mathbf{b} = \mathbf{r}_1 - \mathbf{a} \cdot \mathbf{r}_2$
- **Encryption:** Plaintext $\mathbf{m} \in \{0,1\}^n$ is first encoded as $\overline{\mathbf{m}} \in \mathcal{R}_q$, and then encrypted as
  $$(\mathbf{c}_1, \mathbf{c}_2) = (\mathbf{a} \cdot \mathbf{e}_1 + \mathbf{e}_2, \mathbf{b} \cdot \mathbf{e}_1 + \mathbf{e}_3 + \overline{\mathbf{m}})$$
  with $\mathbf{e}_i \xleftarrow{\chi} \mathcal{R}_q$.
- **Decryption:**
  $$\mathbf{m} = Decode(\mathbf{c}_1 \cdot \mathbf{r}_2 + \mathbf{c}_2)$$
  where *Decode* rounds each coefficient towards $\pm\lfloor q/4 \rfloor$ and then deduces the bits m_i.

In the recent NIST competition[2] for Post Quantum Cryptography standards a number of the candidates were variants of the above cryptosystem based on RLWE. So the discussion in the remaining of the paper could be applied with only a few changes to these cryptosystems. Specifically, the chosen standard Kyber (Bos et al., 2018) for key-encapsulation can benefit from the proposed counter-measure against BP attack.

2.2 Number Theoretic Transform (NTT)

The main operation in RLWE encryption and decryption is the multiplication in $\mathcal{R}_q$. The prime q is generally chosen in order to have a primitive $2n$-th root ω of unity in $\mathcal{R}_q$. In this case, X^n+1 splits entirely and the multiplication in $\mathcal{R}_q$ can be transformed into a point-wise multiplication through the Chinese remainder theorem (CRT) isomorphism

$$\begin{array}{rcl} \mathbb{Z}_q[X]/(X^n+1) & \cong & \prod_{i=0}^{n-1} \mathbb{Z}_q[X]/(X-\omega^{2i+1}) \\ \mathbf{f} & \mapsto & (\mathbf{f}(\omega^{2i+1}))_{i=0,\ldots,n-1} \end{array}$$

where we used the following on the right side: $\mathbf{f}(X) \bmod (X-\omega^{2i+1}) = \mathbf{f}(\omega^{2i+1})$. Then, applying, the above isomorphism to $\mathbf{f}(X) = \sum_{i=0}^{n-1} f_i X^i$ consists in computing for $i = 0, \ldots, n-1$:

$$\mathbf{f}(\omega^{2i+1}) = \sum_{j=0}^{n-1} f_j \omega^{(2i+1)j} = \sum_{j=0}^{n-1} (f_j \omega^j)(\omega^{2i})^j. \quad (1)$$

[2]https://csrc.nist.gov/projects/post-quantum-cryptography

The NTT_n is the multi-evaluation of a polynomial at the n-th roots of unity. This means that the right side of equation (1) is the NTT of $\mathbf{f}'(X) = \sum_{j=0}^{n-1}(f_j\omega^j)X^j$ in the n-th root of unity ω^{2i} for $i = 0,\ldots,n-1$. The NTT_n of a degree $n-1$ polynomial can be computed recursively with $O(n\log(n))$ operations modulo q, as shown in the following subsections.

2.2.1 NTT with Odd-Even Splitting

We briefly recall the approach of Cooley-Tukey (Cooley and Tukey, 1965) for the computation of NTT. This approach uses an even-odd splitting of the polynomial $\mathbf{f}(X)$:

$$\mathbf{f}(X) = \underbrace{\sum_{k=0}^{n/2-1} f_{2i}X^{2i}}_{\mathbf{f}_e(X^2)} + X \times \underbrace{\sum_{i=0}^{n/2-1} f_{2i+1}X^{2i}}_{\mathbf{f}_o(X^2)}$$

If we evaluate the above equation in ω^j for $j = 0,\ldots,n-1$ we will get the following equations. This shows that NTT_n of $\mathbf{f}$ is deduced from $NTT_{n/2}(\mathbf{f}_e)$ and $NTT_{n/2}(\mathbf{f}_o)$ by the following butterfly operations:

$$\begin{aligned}\mathbf{f}(\omega^j) &= \mathbf{f}_e(\omega^{2j}) + \omega^j\mathbf{f}_o(\omega^{2j})\\ \mathbf{f}(\omega^{n/2+j}) &= \mathbf{f}_e(\omega^{2j}) - \omega^j\mathbf{f}_o(\omega^{2j})\end{aligned}$$

Figure 1a shows the sequence of operations (reordering, recursion, multiplication and addition) involved in the Cooley-Tukey NTT computation. We can notice that in this case the multiplication is always done ahead of an addition and a subtraction.

2.2.2 NTT with High-Low Splitting

We recall Gentleman-Sande's approach for the NTT computation. Let us consider the two following polynomials

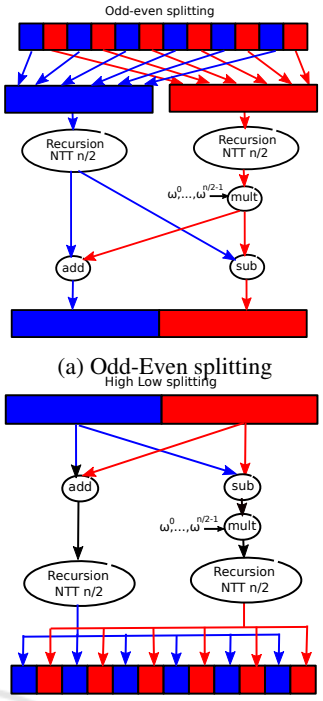
$$\begin{aligned}\mathbf{f}'(X) &= \textstyle\sum_{i=0}^{n/2-1}(f_i + f_{n/2+i})X^i,\\ \mathbf{f}''(X) &= \textstyle\sum_{i=0}^{n/2-1}(f_i - f_{n/2+i})\omega^i X^i.\end{aligned}$$

We obtain the NTT_n coefficients of $\mathbf{f}$ for even powers (resp. odd powers) of ω from the $\text{NTT}_{\frac{n}{2}}$ of $\mathbf{f}'$ (resp. of $\mathbf{f}''$) as follows:

$$\begin{aligned}\mathbf{f}(\omega^{2j}) &= \mathbf{f}'(\omega^{2j}) \text{ for } j = 0,\ldots,\tfrac{n}{2}-1\\ \mathbf{f}(\omega^{2j+1}) &= \mathbf{f}''(\omega^{2j}) \text{ for } j = 0,\ldots,\tfrac{n}{2}-1\end{aligned}$$

The diagram shown in Fig. 1b provides the sequence of operations (High-Low splitting, multiplication, addition/subtraction, recursion and reordering) with the High-Low splitting approach for NTT.

Organisation of the NTT Operations. In the sequel we will randomise the computations done in the NTT, so



(a) Odd-Even splitting

(b) High-Low splitting

Figure 1: NTT recursions.

to clearly explain how these randomisations are done we need to clearly established the sequence of operations done in an NTT. The operations at each level are as follows:

- *In level 0.* We have one group of butterfly operations on the n input coefficients f_i.
- *In level 1.* We have two groups of butterfly operations: one on the coefficients $0,\ldots,n/2$ and a second on the coefficients $0,\ldots,n/2$.
- *In level 2.* we have four groups of butterfly operations, each of size $n/4$.
- etc.

We provide in Fig. 2 an NTT with 2 recursion levels of the High-Low splitting approach.

3 BELIEF PROPAGATION ON NTT

In (Primas et al., 2017) Primas, Pessl and Mangard proposed an attack on NTT which uses side channel leakage of multiplication by the root of unity and apply belief propagation (Pearl, 1982) to deduce most of the coefficients of the output polynomial $NTT(\mathbf{f})$. This attack is an extension of the SASCA attack (Veyrat-Charvillon et al., 2014) on AES to the

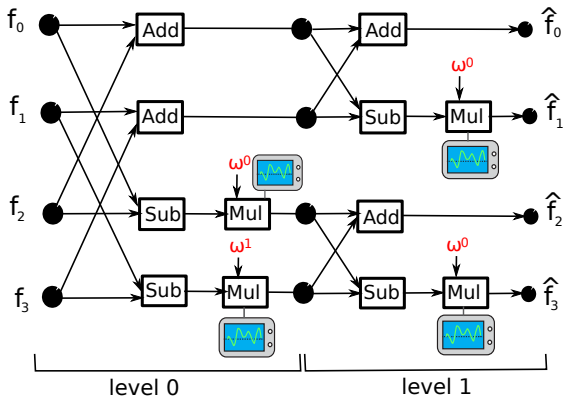


Figure 2: Leakage in NTT_4 computation.

NTT computation. We briefly review the principle of this attack and give some background on belief propagation.

The authors in (Primas et al., 2017) use a leakage in the multiplication by ω^i in the NTT in order to get a likelihood estimation of the value $x \in \mathbb{Z}_q$ processed in this multiplication. Figure 2 shows an example of leakage in NTT_4. This likelihood estimation can be obtained by constructing a template of the leakage of a multiplication by a prescribed root of unity ω^i.

The information provided by all the likelihood estimation at each multiplication of the NTT does not give likelihood estimation for all other values processed in the NTT. Indeed, some temporary values in the NTT are not directly involved in a multiplication with ω^j, so such values remain unknown. The authors in (Primas et al., 2017) propose to use the belief propagation algorithm (BP) (Pearl, 1982) to propagate the scattered information throughout the NTT computations, in order to deduce the missing unknown values.

Let us sketch the idea of belief propagation on a simple example. We consider the addition $z = x + y \bmod q$ of two values $x, y \in \mathbb{Z}_q$. Let us denote $\mathbb{P}(x)$ (resp. $\mathbb{P}(y)$) as the probability (i.e. likelihood estimation) of the value x (resp. y) input to an adder (Fig. 3).

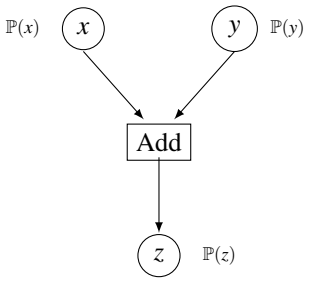


Figure 3: Probability propagation on an addition.

We can propagate these probabilities, to get the probability $\mathbb{P}(z)$ of the output $z = x + y$ as

$$\mathbb{P}(z) = \sum_{x+y \bmod q = z} \mathbb{P}(x)\mathbb{P}(y) \qquad (2)$$

Let us illustrate this fact with a small example: we assume that $q = 3$ and that the probabilities of x and y are the ones shown in the table below. With these probabilities we know with a high probability that $x = -1$ and $y = 0$. Then propagating the probability to z with the above formula leads to the probability of z shown in the table. And then we know that with high probability $z = -1$.

Variable	Probabilities
x	$\mathbb{P}(-1) = 0.9$, $\mathbb{P}(0) = 0.05$, $\mathbb{P}(1) = 0.05$
y	$\mathbb{P}(-1) = 0.03$, $\mathbb{P}(0) = 0.91$, $\mathbb{P}(1) = 0.06$
z	$\mathbb{P}(-1) \cong 0.82$, $\mathbb{P}(0) \cong 0.101$, $\mathbb{P}(1) \cong 0.075$

Note that, in belief propagation, the probabilities are also propagated as follows: from $\mathbb{P}(z)$ and $\mathbb{P}(y)$ to get $\mathbb{P}(x)$ and from $\mathbb{P}(z)$ and $\mathbb{P}(x)$ to get $\mathbb{P}(y)$.

Belief Propagation on NTT. Due to lack of space we just sketch the main process of the Belief Propagation on the NTT of (Primas et al., 2017). For further details the reader may refer to (Primas et al., 2017). For all intermediate variables of the NTT we set a probability for each variable: for variable resulting from a multiplication with ω^k this probability comes from a template of the power consumption leakage, for other variables the probality it set to the uniform distribution. The propagation is then done by applying propagation formula similar to the one in (2) for each Add and Sub operation of the NTT. We repeat this process a number of times, and this effectively propagates the information of the templates to unkown variables (the one with uniform distribution). At the end, we succeed if in the output variables of the NTT the values with the highest propbabilities a the correct correct values.

4 RANDOMISATION OF NTT

In this section we first review the randomisation proposed in the literature. We then present a set of randomisations and evaluate their effect to prevent Belief Propagation attack on NTT.

State of the Art: Randomisation at the Input of the NTT. In the literature we can find the three following strategies for the randomisation of the input of the NTT:

Multiplicative mask	$\mathbf{f}' = \alpha \times \mathbf{f}$ with a random $\alpha \in \mathbb{Z}_q$ $\widehat{\mathbf{f}'} = NTT(\mathbf{f}')$ $\widehat{\mathbf{f}} = \alpha^{-1} \times \widehat{\mathbf{f}'}$
Additive mask	random split $\mathbf{f} = \mathbf{f}' + \mathbf{f}''$ $\widehat{\mathbf{f}'} = NTT(\mathbf{f}')$ $\widehat{\mathbf{f}''} = NTT(\mathbf{f}'')$ $\widehat{\mathbf{f}} = \widehat{\mathbf{f}'} + \widehat{\mathbf{f}''}$
Shifting	$\mathbf{f}' = X^r \times \mathbf{f} \mod (X^n - 1)$ $\widehat{\mathbf{f}'} = NTT(\mathbf{f}')$ $\widehat{\mathbf{f}} = (\omega^{-ir}\widehat{\mathbf{f}'}_i)_{i=0,\ldots,n-1}$

These three randomisations do not protect NTT from the attack of Primas *et al.* based on Belief Propagation. Indeed, the attack can be conducted on the computation of $NTT(\mathbf{f}')$ (and $NTT(\mathbf{f}'')$ for the additive mask). Then to recover $\mathbf{f}$ we just have to add $\mathbf{f}'$ and $\mathbf{f}''$ for the additive mask, or guess the random parameter (α or r) for the two other strategies and then proceed to the computation of the secret key as in Primas *et al.* attack (Primas et al., 2017).

State of the Art: Randomisation of the Whole NTT.

Ravi *et al.* in (Ravi et al., 2020) proposed the following shuffling and masking randomisation to protect the NTT from Primas *et al.*'s attack..

Shuffling. This approach is detailed in (Ravi et al., 2020) but it is classical approach known for some time. At each level butterfly operations are performed in a random order, using a pseudo random generator which randomly produces the pairs of (m,i) of each butterfly iteration of level j: m gives the group of butterflies of level j and i the index of this butterfly in this group.

This randomisation renders difficult to associate each portion of power trace of multiplication by a root ω^k of unity. But if the processed values are not masked, it could then be easier for attacker to find the root ω^k involved in the trace and/or link input and output value from two NTT recursion levels. So this randomisation might be associated with another one to be effective.

Let us eavaluation the level of randomisation: in the general case, at each level of NTT the are $\frac{n}{2}!$ possible reordering of the pairs (m,i). This gives a level of randomization of $(\frac{n}{2}!)^{\log_2(n)}$.

Multiplicative Mask. This approach masks each intermediate variable a with a multiplicative mask $a' = a \times \omega^{r_a}$, with r_a random, which is refreshed at each butterfly operation. In (Ravi et al., 2020) the authors propose several variants of this approach to reduce the cost. But all these strategies have, at least, an increase in cost of $\frac{n}{2}\log_2(n)$ multiplications in the NTT.

4.1 Proposed Randomisations

We present in this section a set of virtually free randomisation techniques of the NTT. Virtually free means that they do not induce additionnal operations in $\mathbb{Z}_q$. Shuffling approach is an example of such approach, and we will consider it in the sequel as a complement of the proposed approaches.

4.1.1 Random Primitive Root of Unity

- *Description of the randomisation.* This approach only applies to NTT with High-Low splitting. In this approach, at any recursive level of the NTT, and before each group of butterfly operations, we randomly pick the primitive root of unity. Since $n = 2^t$ the primitive roots are w^e with e an odd integer smaller than n. This results in the following pseudo-code for the splitting part of the NTT recursion.

  $e \xleftarrow{\mathcal{U}} \{1,3,\ldots,n-3,n-1\}$
  $\omega' = \omega^e$
  // High-Low splitting and butterfly operations
  $\mathbf{f}' \leftarrow \sum_{i=0}^{n/2-1}(f_i + f_{n/2+i})X^i$
  $\mathbf{f}'' \leftarrow \sum_{i=0}^{n/2-1}(f_i - f_{n/2+i})\omega'^i X^i$

  This randomisation does not involve any additional cost in $\mathbb{Z}_q$, when all ω^i are precomputed.

- *Quality of the Randomisation.* As we can see in the pseudo code
  - At each recursion level, only half of the coefficients are masked by the random root: $n/2$ in level 1, $2 \times n/4 = n/2$ in level 2, $4 \times n/8 = n/2$ in level 3, and so on.
  - The number of coefficients masked at least once during the whole NTT computation is $n/2$ after the level 1, $n/2 + n/4$ after level 2, $n/2 + n/4 + n/8$ after level 3 and so on. So at the end we get $n-2$ masked coefficients (the last level does not involve multiplication).
- *Level of Randomisation* At the i-th level of the NTT recursion, there are 2^i group of butterfly operation and for each group we pick ω' among $n/2^{i+1}$ primitive roots of unity. Taking the product for $i = 0,\ldots,t-2$, we obtain the following level of randomisation:

$$\prod_{i=0}^{t-2}(2^{t-1-i})^{2^i} = 2^{\Sigma_{i=0}^{t-2}(t-1-i)2^i} = \frac{2^{n-1}}{n}$$

4.1.2 Random Multiplicative Mask in Each Multiplication with a Root of Unity

This randomisation applies only on NTT formula with High-Low splitting.

- *Description of the Randomisation.* In this approach, before the High-Low butterfly operation of a recursion level we pick a random r, and we replace in the multiplication ω^i by $\omega^i \times \omega^r = \omega^{i+r}$. The resulting randomised butterfly operations are shown below :

  $r \xleftarrow{\mathcal{U}} \{0,\ldots,n-1\}$
  // HL splitting and butterfly operation
  $\mathbf{f}' \leftarrow \sum_{i=0}^{n/2-1}(f_i + f_{n/2+i})X^i$
  $\mathbf{f}'' \leftarrow \sum_{i=0}^{n/2-1}(f_i - f_{n/2+i})\omega^{i+r}X^i$

  Since all power of ω are all precomputed this does not require any additional computation, the resulting coefficients of $NTT(\mathbf{f}'')$ are masked by ω^r.

  This approach is somehow a variation of the multiplicative mask randomisation of (Ravi et al., 2020): we apply it on High-Low splitting formula and with a and b sharing the same mask.

- *Quality of Randomisation.* The analysis is exactly the same as in the randomisation of the primitive root of unity. This means that at each NTT level $n/2$ values are masked, and at the end of the NTT $n-2$ coefficients are masked at least once during the computation.

  This version of the randomisation with multiplicative mask, is weaker than the one of Ravi *et al.* (Ravi et al., 2020). Indeed in (Ravi et al., 2020) all variables are masked during the whole NTT computation. But, in counter part, the proposed approach is more efficient since it does not involved any additional cost.

- *Level of Randomisation.* The analysis of the randomisation shows that: at recursive level 1 we pick one random r among n integers, at level 2 we pick two random r among n integers, at level 3 we pick four random r among n integers and so on until level $t-2$. This results in the following level of randomisation:

$$\prod_{i=0}^{t-2}(n)^{2^i} = n^{\frac{n}{2}-1}$$

4.1.3 Random High-Low/Odd-Even Butterfly Operations

We randomise the sequence of butterfly operations of the NTT by randomly choosing the type of splitting used (i.e. High/Low or Odd/Even) at each group of each level.

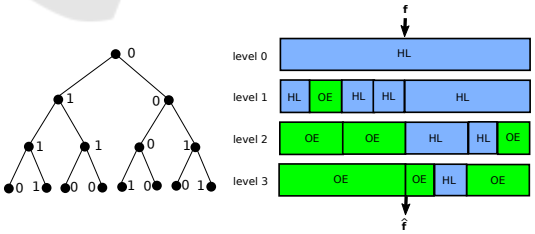
- *Description of the randomisation.* In Section 2 we reviewed the two approaches for the recursive computation of NTT: High-Low and Odd-Even splitting. We also showed that the sequence of operations (addition, subtraction, multiplication and recursion) in these two approaches are different. We propose to randomly choose High-Low or Odd-Even splitting recursion for each group of butterfly operations.

  A sketch of the pseudo code of this approach is shown below:

  $r \xleftarrow{\mathcal{U}} \{0,1\}$
  if $r = 0$ **then**
  // Apply High-Low splitting butterfly and recursion
  ...
  else
  // Apply Odd-Even splitting butterfly and recursion
  ...

- *Quality of Randomisation.* Since the sequence of operations is random, the intermediate values in the NTT computation are also random and then less predictable. But the sequence of operations could leak out the type of splitting approach used (i.e. the values of the random bits r): addition/subtraction followed by a multiplication corresponds to a High-Low splitting butterfly, while multiplication followed by addition/subtraction corresponds to an Odd-Even splitting butterfly.

- *Example.* We set $n = 16$ and we pick the values of r shown in Subfig 4a for each recursive level.



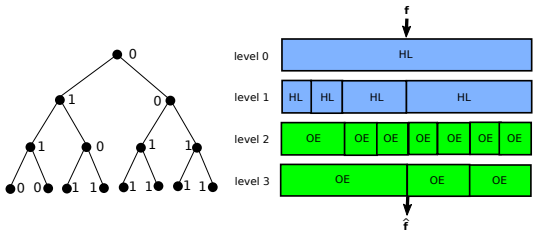
(a) Random bits (b) Butterfly operations

Figure 4: Leakage in HLOE randomisation.

The diagram in Subfig 4b shows the organisation of the operation for the above random bits r. We can see that the block with OE butterfly operations in level 1 is leaked out, as is the block with HL butterfly operations in the level 3.

In order to prevent the leakage due to the sequence of operations, we can restrict the bit r to be half of

the time 0 and half of the time 1 in each path from root to leaves in the recursion tree. In Figure 5 we show an example on such restricted randomization.



(a) Balanced random bits (b) Non-leaking butterfly operations

Figure 5: Non leaking HLOE randomisation.

- *Level of Randomisation.* If we can pick freely the bit r for each group of butterfly operations, we would get the following level of randomisation :

$$1+2+2^2+\ldots+2^{t-2}=\frac{n}{2}-1$$

But, to prevent leakage, if we restrict the randomisation in order to get half of the levels with High-Low butterfly operations and the other half with Odd-Even butterfly operations, then only for the first $t/2$ level the bit r can be chosen freely. This means that the level of randomisation is at least :

$$1+2+2^2+\ldots+2^{t/2}\cong\sqrt{n}$$

4.1.4 Randomise Modular Reduction

We propose to randomly select the algorithm for multiplication modulo q for each group of butterfly operations.

- *Description of the Randomisation.* There are several ways to perform a multiplication of two elements x and y modulo q, the two most popular are the Montgomery multiplication and the Barrett multiplication. Let us assume that q is an ℓ-bit prime integer. In Barrett algorithm reduces the product $x\times y$ by clearing the ℓ most significant bits, while Montgomery algorithm reduces the product $x\times y$ by clearing the ℓ least significant bif of $x\times y$, producing a factor $2^{-\ell}$ in the result. then these two algorithm are shown in Table 1.

Table 1: Barrett and Montgomery modular multiplication.

Barrett modular mult	Montgomery modular mult
Precomp. $q'=\lfloor 2^{2\ell}/q\rfloor$	Precomp. $q'=q^{-1} \mod 2^{\ell}$
$z\leftarrow x\times y$ $s\leftarrow \lfloor\lfloor z/2^{\ell-1}\rfloor q'/2^{\ell+1}\rfloor$ $r\leftarrow z-s\times q$	$z\leftarrow x\times y$ $s\leftarrow q'\times z \pmod{2^{\ell}}$ $r\leftarrow (z-s\times q)/2^{\ell}$

Then we modify the High-Low splitting approach of NTT as follows: before each step of butterfly operation we flip a coin b and we apply Barrett multiplication (BM) if $b=0$ and Montgomery multiplication if $b=1$. A sketch of the pseudocode for randomised butterfly operations is shown below :

$b \xleftarrow{\mathcal{U}} \{0,1\}$
if $b=0$ **then**
// High-Low butterfly with BM and recursion
...
else
// High-Low butterfly with MM and recursion
...

- *Quality of the Randomisation.* We notice the following:
  - Only the values involved in a multiplication with ω^i are randomised. This means that $n/2$ value are randomised at each level of recursion and at the end, $n-2$ values are randomised at least once during the NTT.
  - A Montgomery multiplication produces a multiplicative mask $2^{-\ell}$. A final coefficient $\hat{f'}_i$ is involved between 0 and $t-2$ multiplications in the whole randomised NTT computation. This means that $\hat{f}'_i$ has a random multiplicative mask as follows

$$\hat{f}'_i=\hat{f}_i\times(2^{-\ell})^k \text{ for some } k\in\{0,\ldots,t-2\}.$$

  - Even if Barrett multiplication and Montgomery multiplication are very similar in terms of the sequence of operations (multiplications and shiftings), there are some slight differences (the type of shift done). These slight differences could be visible in the power consumption leakage and this could leak out which algorithm was used, revealing the randomisation used.

- *Level of Randomisation.* For $n=2^t$, at the first level of recursion we flip one coin, at level 2 we flip two coins, at third level we flip 4 coins and so on. In the last level there is no multiplication, so randomisation stop at level $t-1$. We then obtain the following level of randomisation:

$$1+2+2^2+\ldots+2^{t-2}=\frac{n}{2}-1.$$

5 SECURITY EVALUATION OF THE PROPOSED RANDOMISATIONS OF NTT

In this section we present simulation results of Belief Propagation. We first show how the Belief Propagation behaves when no randomisation is applied in the NTT. Afterwards for each proposed randomisation we will show how the Belief Propagation is affected and finally we will study some combination of randomisation to see if it reinforces the security of the implementation.

Leakage of Multiplication. To simulate the power consumption we used the Hamming distance between the considered value v and the processed value v_p without any noise. So the attacker has a good information on the value processed in a multiplication. We derive the probability that a value v is involved in the multiplication $v_p \times \omega^i \mod q$ with a centered Gaussian likelihood estimator with deviation $\sigma = 0.25$:

$$\mathbb{P}(v) = e^{-\frac{HamDist(v,v_p)^2}{\sigma}}.$$

Note that this assumption gives more power to the attacker than the model used in (Primas et al., 2017). Indeed, in (Primas et al., 2017) the leakage model the hamming weight of the processed value which leaks out less information that the hamming distance.

5.1 Simulation Results for Non-Mixed Randomisations

Now we present simulation results of Belief Propagation on NTT for different randomisation approaches and for $n = 256$ and $q = 3329$, which are the ones used in Kyber (Bos et al., 2018).

We consider in this section randomisation of NTT with only one approach among (RandRoot, MulMask, RandRed and RandHLOE) and the Shuffling randomisation of Ravi (Ravi et al., 2020). For comparison purpose, we also provide the result for a non-randomised NTT.

5.1.1 Correct Guess in the Whole NTT

Fig. 6 shows the number of values with highest probability which are correct among all variable nodes of Belief Propagation graph. For $n = 2^t$ the total number of variable nodes are $(t+1) \times n$, so for $n = 256$ we have 2356 variable nodes.

For the non-randomised NTT, the curve shows that we need around t iterations to reach a maximum of 1800 variables with correctly guessed values (among 2356 variables). The other variables are hidden by the remaining uniform probability sources, and we believe that this cannot be changed by more BP iterations. So we cannot recover more variables.

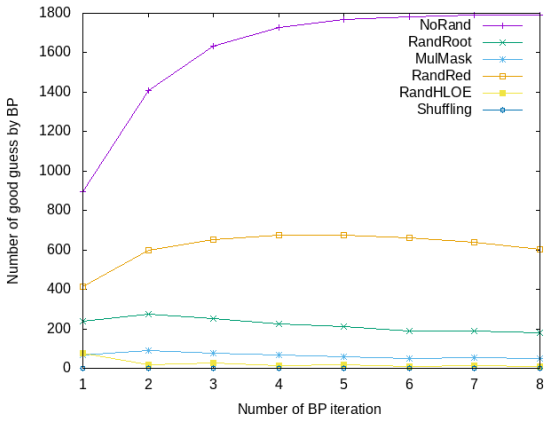


Figure 6: Number of correct coefficient on the whole NTT.

For the randomised NTTs, we can notice that the number of correct guess is significantly higher for randomisation which does not affect the sequence of operations in NTT (i.e. RandRoot, RandMultMask, RandRed). We can also notice that RandRed is the least effective randomisation.

The HLOE randomisation, disorganises the operations done in NTT, so the graph used in Belief propagation does not match the sequence of operations processed for the NTT and this produces the low number of correct guess. The same is true for the Shuffling approach of (Ravi et al., 2020).

5.1.2 Correct Guess in the Last Level of NTT

Looking at the correct guess in the last level tells us if the Belief Propagation is inefficient or not. Indeed, if the number of correct guess is really small, then almost no information leaks out and the secret cannot be recovered. In Fig. 7 we show the number of correct guess for non-randomised NTT and for all considered randomisation during the execution of Belief Propagation.

We first notice that, in the case of non-randomised NTT, the number of correct guess increases during the first iterations, to reach the maximal value of $254 = n - 2$. For the randomised NTT most of the curves are flat or decrease. The only curve which increases is the one for RandRed, which means that belief propagation is able to combine distant information to recover some unknown values. This means that the leakage in RandRed approach remains important.

But we can notice that all the considered randomisations prevent the Belief Propagation to be success-

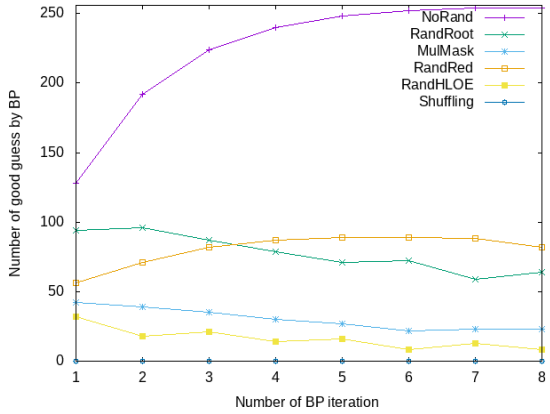


Figure 7: Correct guessed value in last NTT level.

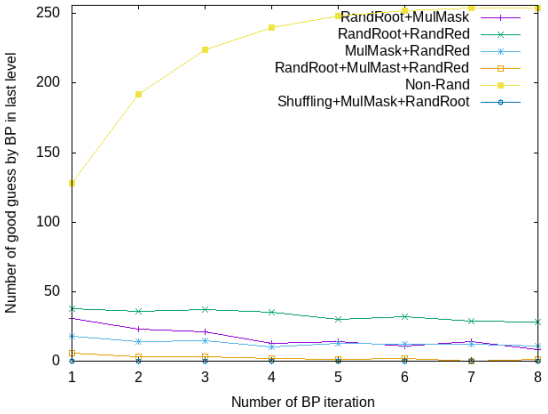


Figure 8: Total number of correct guess per BP iteration for combined randomisations.

ful: the number of correctly guess coefficients is too small to recover the whole secret output.

5.2 Simulation Results for Mixed Randomisation

Some randomisations can be combined: RandOrder, RandRed, RandRoot and RandMulMask. Indeed they are all applied to NTT with High-Low splitting strategy and the randomisations can be applied independently from each other.

The RandHLOE approach partly uses Odd-Even splitting formula, which requires to have data with the same multiplicative mask. This prevents the use of RandHLOE combined with RandRed, RandRoot and/or RandMulMask since they produces masked data with possibly different masks.

The shuffling randomisation of (Ravi et al., 2020) can be combined to all the other approaches. But since shuffling already provide curve really close to zero, we only provide the simulation result for Shuffling+RandRoot+RandMulMask.

The numbers show that, as expected, the combination of two or three randomisations lower the number of correct guess. The combination RandRoot+MulMask+RandRed give a curve really close to zero, this means that it does not leak any information.

The combination of RandRoot+RandMulMask+Shuffling seems to be the most effective one. Indeed, even if the attacker is able to know the root of unity used in each multiplication, the use of the RandRoot prevent him to know the node of graph of the NTT it should be associated with: it could be anyone in the same group. And in this case, the graph cannot be correctly constructed and Belief Propagation would then fails.

6 CONCLUSION

In this paper we considered the security of Ring LWE cryptosystems relatively to Primas *et al.*'s attack based on Belief Propagation (Primas et al., 2017). The attack in (Primas et al., 2017) focuses on the Number Theoretic Transform and exploits scattered leakage information to recover the output of NTT. We proposed a set of virtually free randomisations which reorganise/modify the sequence of operations of the NTT and/or induce a random mask in the processed values.

The level of randomisation of each approach is shown in Table 2. These results show that the level of randomisation for RandHLOE and RandRed is really low and could be attacked by exhaustive search of the random bits used. The other randomisations have a quite large level of randomisation.

Table 2: Level of randomisation.

	Formula	$n = 256$
Shuffling (Ravi et al., 2020)	$(\frac{n}{2}!)^{\log_2(n)}$	2^{5729}
RandHLOE	$\sim \sqrt{n}$	2^7
RandRoot	$\sim \frac{2^n}{n}$	2^{247}
RandMulMask	$\sim n^{n/2}$	2^{1024}
RandRed	$n/2$	2^7

We applied Belief Propagation on all randomisations of NTT and also on some combination of these randomisations. These simulation results told us that Shuffling, RandHLOE are the most efficient to prevent Belief Propagation, since the number of correctly guessed values with BP remains low.

Finally, when considering combination of randomisations, we noticed that RandHLOE cannot be combined with RandRoot, RandMulMask and/or RandRed. The best approach combines Shuffling,

RandRoot and RandMulMask which has a high level of randomisation, and even if the roots of unity can be guessed from leakage they cannot be used to apply the attack since it is not possible to construct the Belief Propagation graph of NTT.

REFERENCES

Bos, J. W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2018). CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE.

Cooley, J. and Tukey, J. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301.

Lyubashevsky, V., Peikert, C., and Regev, O. (2010). On Ideal Lattices and Learning with Errors over Rings. In *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer.

Pearl, J. (1982). Reverend bayes on inference engines: A distributed hierarchical approach. *Probabilistic and Causal Inference*.

Primas, R., Pessl, P., and Mangard, S. (2017). Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. In *CHES 2017*, volume 10529 of *LNCS*, pages 513–533. Springer.

Ravi, P., Poussier, R., Bhasin, S., and Chattopadhyay, A. (2020). On configurable SCA countermeasures against single trace attacks for the NTT - A performance evaluation study over kyber and dilithium on the ARM cortex-m4. In *SPACE 2020*, volume 12586 of *LNCS*, pages 123–146. Springer.

Regev, O. (2009). On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40.

Shor, P. (1999). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Rev.*, 41(2):303–332.

Veyrat-Charvillon, N., Gérard, B., and Standaert, F. (2014). Soft analytical side-channel attacks. In *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 282–296. Springer.