Fault analysis of the Number Theoretic Transform: Case study on three Lattice based NIST PQC candidates

No Author Given

No Institute Given

Abstract. The recent NIST standardization process for post-quantum cryptography has attracted many proposals, with the majority of them based on lattice-based cryptography. Many of the lattice-based proposals, such as `KYBER`, `DILITHIUM`, and `NEWHOPE` which based on the more structured versions of the Learning With Errors (LWE) problem that is; Ring-LWE or Module-LWE enjoy equally good security guarantees and efficiency guarantees. These schemes which compute over polynomials in rings contain in their core, the Number Theoretic Transform (NTT) which is mainly used for fast polynomial multiplication. Thus NTT is inherently a target for side-channel analysis, especially since it is used for computation over the secret-key information. In this paper, we investigate the fault vulnerability and fault propogation characteristics of the NTT transform, with specific focus to the aforementioned NIST candidates. We identify a number of vulnerabilities in the NTT operation that enables us to successfully perform valid key and message recovery attacks over the `NEWHOPE` proposal. We are also able to find specific scenarios where all the three schemes are vulnerable to practical key recovery and message recovery attacks through faulting the NTT operation. We found that the NTT operation using the GS butterfly is more susceptible to fault attacks compared to the one using the CT-butterfly due to the the specific structure of the GS-butterfly. We back our claims with concrete theoretical, simulation and experimental results using practical fault models. The results are highly relevant to the NIST post-quantum standardization effort, showing that practical fault analysis is possible, and applicable, for any lattice-based cryptographic scheme implementing the NTT.

1 Introduction

The National Institute for Standards and Technology (NIST) have recently begun a standardisation effort for post-quantum cryptography [24], similar to previous standardisations of AES and SHA-3. This initiative is due to the emerging threat of quantum computers [18] to classical public-key cryptographic primitives, which provide the essential security services to almost all known digital infrastructures. This threat is mainly due to Shor's [36] and Grover's [16] algorithms, which strengthen cryptanalysis against currently used cryptographic systems based on the hardness of factoring (RSA), discrete logarithms (ECC/ECDSA), and symmetric-key primitives (AES/SHA-3).

In addition to evaluating the security of all the proposals in terms of both classical and quantum cryptanalysis, NIST also stresses on the need for secure implementations of post-quantum cryptographic algorithms over a variety of devices, ranging from high-end server systems to low-end microcontrollers used in embedded devices. This directly implies the need to thoroughly study the implementation security aspects of all the proposals against possible passive, and active, side-channel attacks.

Amongst all the NIST proposals submitted for standardization, lattice-based cryptography has emerged as the largest category in terms of the total numbers. In the last decade, there has been a significant amount of research in the field of lattice-based cryptography, covering many aspects of cryptography, including strong theoretical foundations [21, 23] practical implementations [26, 28], classical and quantum cryptanalyis [9,12], and detailed research on implementation security aspects [14].

Of particular interest are lattice-based schemes which base their hardness on a structured version of the Learning with Errors (LWE) problem, like Ring-LWE and Module-LWE [20, 23]. These schemes deal with computation over polynomials in a ring with special algebraic properties. Amongst the operations performed in these schemes, the polynomial multiplication operation is usually the most expensive, both in terms of computational resources and time, but has been made efficiently computable thanks to the introduction of the Number Theoretic Transform (NTT) operation. The NTT operation yields quasi-linear time complexity ($\mathcal{O}(n\log(n))$ for the polynomial multiplication operation, which usually has quadratic run-time ($\mathcal{O}(n^2)$) for computation. Thus, the NTT operation serves as an important building block for most, if not all, lattice-based schemes based on these structured versions of LWE. This is especially true considering that secret-key information is usually involved in these NTT multiplications, which are an obvious target for side-channel analysis.

In this paper, we target the NTT multiplier with respect to fault analysis, and consider our attack on schemes submitted for NIST post-quantum standardization, these being; the `NEWHOPE` [27] and `KYBER` [35] key encapsulation mechanisms and the `DILITHIUM` [22] signature scheme. To the best of our knowledge, we perform a first of its kind study of the NTT operation's faulty vulnerability and fault propagation characteristics. We show that the NTT operation can be a very susceptible target for fault analysis, and show that secret information can be derived via certain attack scenarios. All attacks assume a very realistic and practical fault model (instruction skip), that can lead to message recovery and key recovery attack over the aforementioned lattice-based cryptographic schemes.

1.1 Related work

Previous research on fault analysis in lattice-based cryptography has solely focused on digital signature schemes. The first analysis on various lattice-based signature schemes was proposed by Bindel *et al.* [5], focusing on GLP [17], BLISS [11], and Ring-Tesla [2] schemes that follow the Fiat-Shamir framework. While most of

the attacks were based on realistic fault model assumptions, the others required very high number of faults, making them difficult to reproduce in practice.

Espitau *et al.* [13] aimed at Fiat-Shamir and hash-and-sign type signature schemes. They developed attacks utilizing loop abort faults, showing that the signature generation step can be converted into a solvable closest vector problem (CVP) instance. This was due to the ability of loop abort faults to limit the masking polynomials to low degrees. Bindel *et al.* [6] later proposed an overview of countermeasures for the existing fault attacks.

Away from lattice-based cryptography, there are also works which report different fault attack models that were achieved in a practical settings, using various fault injection techniques. Laser has been used for precise instruction skips, enabling to efficiently break implementations of AES [8] and ChaCha-20 [19]. Similar models were achieved by clock glitches, which can be introduced by inexpensive equipment [7]. Electromagnetic fault injection was shown to be effective in tampering with instruction cache, leading to skipping or replaying instructions precisely [31].

1.2 Our Contributions

1. To the best of our knowledge, we perform the first of its kind analysis of the fault vulnerability and fault propogation characteristics of the NTT operation.
2. We identify a unique property called *diffusion* in the NTT operation. We propose to inject faults to stifle diffusion in the NTT operation. The corresponding faulty LWE instance utilizing these faulted NTT operations can be broken down into smaller independent ISIS instances which can be trivially solved by the attacker to retrieve the secret.
3. We also identify another attack especially over the NTT operation using the GS-butterfly which when faulted in a certain way can lead to secrets with very low entropy which are easier to solve.
4. We analyse the various implementations of the NTT operation used in reference implementations of the considered NIST PQC candidates and show how these vulnerabilities can be exploited to lead to practical message recovery and key recovery attacks in certain scenarios.
5. We provide both theoretical and practical results of the attack complexity and the number of faults to be injected for all the paramter sets of the considered schemes.
6. We also show practical experiments results of laser fault injection on the AVR micro-controller. We were able to achieve a 100% repeatability for each of the assumed fault model in this paper, thus showing the practicality of our identified fault vulnerability in the NTT transform.

2 Preliminaries

Notation. For a prime number q, we denote by $\mathbb{Z}_q$ the field of integers modulo q. For a positive integer n, we denote by $\mathcal{R} = \mathbb{Z}_q[x]/(x^n + 1)$ the quotient ring

obtained by reducing polynomials in $\mathbb{Z}_q[x]$ modulo $x^n + 1$ and coefficients modulo q. We express polynomials by their vector of coefficients, which we denote by small bold letters, e.g. $\mathbf{x} \in \mathbb{Z}_q^n$. We say these are in *normal domain*. Furthermore we denote the transformation of $\mathbf{x}$ by means of the NTT by $\hat{\mathbf{x}}$, which we say is in *NTT domain*. Variables in either domain can be faulted, which we denote by $\mathbf{x}^*$ or $\hat{\mathbf{x}}^*$, respectively.

Throughout the paper we will need to use several different types of multiplications. For two polynomials $\mathbf{x}$ and $\mathbf{y}$, we denote by $\mathbf{x} \times \mathbf{y}$ their multiplication in normal domain and by $\hat{\mathbf{x}} \circ \hat{\mathbf{y}}$ their component-wise multiplication in NTT domain. Scalar multiplication is denoted by $x \cdot y$ for $x, y \in \mathbb{Z}_q$.

Mathematical problems. Our work is relevant for the analysis of those variants of the LWE problem [30] where the NTT is used. This includes Ring-LWE [23] and Module-LWE [20] problems. The core idea of all of them is similar: a public operand (matrix, polynomial or matrix of polynomials) is multiplied by a secret operand. A small error (i.e., noise) is added, and the final output is returned. By "small", we mean that values are taken from an interval much smaller than $\mathbb{Z}_q$. Also, variants of LWE exist where this error is either uniform or discrete Gaussian distributed. Either way, we will simply refer to the range of possible values the error (and sometimes also the secret) can take by $[-\eta, +\eta]$, without specifying the actual distribution.

The search variant of each problem asks to find the secret operand, while the decision variant asks to distinguish the output from a random value uniformly drawn from the space over which each problem is defined. Our analysis is fairly oblivious to all technicalities beyond such a high level description, hence we refer to the original papers for more details.

2.1 Number Theoretic Transform

The NTT based polynomial multiplier is used extensively within lattice-based schemes. The main reason is that polynomial multiplication operations can be performed very efficiently by first applying the NTT, performing a component-wise multiplication, and finally apply the Inverse NTT (INTT).

The NTT transformation is a bijective mapping from one sequence into another sequence of the same length. Since we deal with integers in $[0, q-1]$, we can define the NTT function as follows:

$$\mathrm{NTT} : \mathbb{Z}_q^n \to \mathbb{Z}_q^n$$

such that the sequence $\mathbf{p}$ with n elements $(\mathbf{p}[0], \ldots, \mathbf{p}[n-1])$ is mapped to $\hat{\mathbf{p}}$ as

$$\hat{\mathbf{p}}[j] = \sum_{i=0}^{n-1} \mathbf{p}[i] \cdot \omega^{i \cdot j} \tag{1}$$

where $j \in [0, n-1]$ and ω being the n^{th} root of unity in the operating ring $\mathbb{Z}_q$.

2.1.1 NTT Restrictions The use of NTT for polynomial multiplication requires a mandatory choice of an NTT-friendly polynomial for the ring's modulo polynomial (which in our case is $X^n + 1$) and also an NTT-friendly prime which satisfies the condition $q \equiv 1 \mod 2n$ with n being a power of 2. In order to perform the polynomial multiplication using the negative wrapped convolution, it requires the presence of both the n^{th} and $2n^{th}$ roots of unity in the ring, which is ensured by the above mentioned choice of these parameters.

2.1.2 NTT usage An NTT module [25] achieves high speed for polynomial-multiplication, namely quasi-linear runtime with $\mathcal{O}(n \log(n))$ multiplications in $\mathbb{Z}_q$, and is used to implement the equivalent negative wrapped convolution. This is the main reason for its ubiquitous use within lattice-based cryptography, as using conventional schoolbook multiplication would yield quadratic runtime $\mathcal{O}(n^2)$. Effectively, the product of two polynomials $\mathbf{x}$ and $\mathbf{y}$ is computed as

$$\mathbf{z} = \text{INTT}(\text{NTT}(\mathbf{x}) \circ \text{NTT}(\mathbf{y})).$$

The multiplication in the NTT domain, denoted by $\circ$, is component-wise and can then be computed in $\mathcal{O}(n)$.

2.1.3 NTT Structure The NTT transformation of a sequence with composite size n can be recursively broken down into p smaller NTT transformations, each of size m such that $n = p \times m$. These smaller NTT transformations can be further broken down into atomic operations called butterfly operations, which themselves are NTT transformations of size r, the radix of the transformation. Lattice-based schemes usually adopt the radix-2 NTT transformation and thus deal with butterfly operations of size 2. The whole NTT operation is implemented as butterfly operations spanning over multiple stages.

The NTT transformation for a given n point sequence consists of $\log(n)$ stages with each stage consisting of $n/2$ butterfly operations. A radix-2 butterfly operation takes two inputs $(a, b) \in \mathbb{Z}_q^2$ along with a given constant w and produces two corresponding outputs $(c, d) \in \mathbb{Z}_q^2$. The two inputs (a, b) are elements of the sequence, while w is called the twiddle factor, which is either a power of the n^{th} root of unity (ω^i for $0 < i < n-1$) or $2n^{th}$ root of unity (ψ^i for $0 < i < 2n-1$). The sequence of inputs provided to the butterfly operations, and the value of the twiddle factors used varies, depending on the stage of the NTT transformation.

There are two types of butterfly operations, the Cooley-Tukey (CT) butterfly [10] and the Gentleman-Sande (GS) butterfly [15]. Since both can be interchangeably used to perform both the NTT and the INTT operations and all the analysis covered in this paper equally applies to both the CT and GS butterfly (except for a specific type of analyis that only applies to the the GS butterfly in Section 5), thus for the remaining of this work we focus on the former, whose structure is:

$$\begin{aligned} c &= a + b \cdot w \\ d &= a - b \cdot w, \end{aligned} \tag{2}$$

for inputs (a, b) and an associated twiddle factor w. We refer to Figure 1a which illustrates a signal flow graph of an example 8-point NTT operation.

Apart from the transformations, the NTT polynomial multiplier also has a number of overhead steps, like pre-scaling (multiplication with powers of ψ), post-scaling (multiplication with powers of ψ^{-1}) and bit-reversal of the order of the inputs, which adds a significant overhead to the polynomial multiplication operation. Roy et al. [32] and Pöppelmann et al. [29] published subsequent works to propose modified NTT algorithms to remove all the aforementioned overhead steps of the NTT based polynomial multiplier.

2.2 Inhomogeneous Short Integer Solution Problem (ISIS)

The ISIS problem is a lattice problem with relation to linear systems of equations. It is also considered as the vectorised version of the famous *knapsack problem*. The problem is defined as: given a modulus q, a small set $\mathcal{B} \subset \mathbb{Z}$ that contains 0, an $n \times m$ matrix A, such that $m > n$ and a column vector $s \in Z_q^n$, find a column vector $x \in \mathcal{B}^m$ such that

$$A \cdot x \equiv s \pmod{q} \tag{3}$$

The ISIS problem has been heavily studied and an interested reader may refer to the recent paper [4], discussing different combinatorial methods of solving the problem. However, in this paper we are only interested in some of the properties of the problem. While Equation 3 can be viewed as an under-defined system of linear equations with q^{m-n} solutions, there is no guarantee that any of these solutions exist in $\mathcal{B}^m$. Hence, an ISIS instance can have no solutions, a unique solution, or many solutions. For the this reason, the ISIS density parameter is introduced.

The density of an $(m, n, q, \mathcal{B})$-ISIS instance, $\delta = \frac{|\mathcal{B}|^m}{q^n}$. The density of an ISIS instance can have one of two different interpretations:

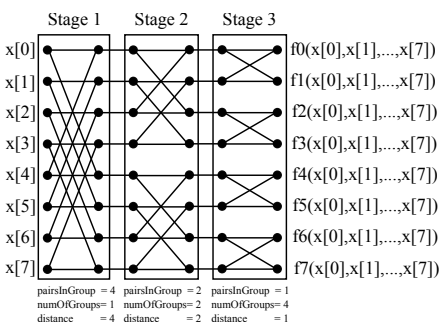
1. $\delta > 1$: in this case it represents the average number of solutions for a random choice of A and s.
2. $\delta \leq 1$: in this case it represents the probability of finding at least 1 solution for a random choice of A and s.

For the purposes of this paper, we will consider cases where $\delta \ll 1$, which are instances of very low probability of finding any solution. Hence, we estimate the number of solutions of such instances to be close to 1, and much smaller than $|\mathcal{B}|^m$. The smaller δ is, the more our assumption is close to reality. Previous research [34] has shown algorithms for solving hard knapsack and ISIS problems, with complexities of $\mathcal{O}(2^{m/2})$ and below.

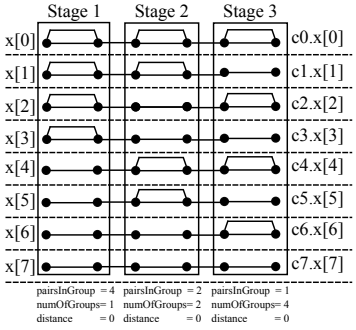
3 Fault analysis of NTT operation

Referring to Equation 1, which defines the NTT operation, we can see that every element of the NTT output $\hat{\mathbf{p}}$ can be alternatively represented as:

$$\hat{\mathbf{p}}[j] = f_j(\mathbf{p}[0], \mathbf{p}[1], \mathbf{p}[2], \ldots, \mathbf{p}[n-1]), \tag{4}$$



(a) Non-faulted NTT according to code from Algorithm 1.



(b) Faulted NTT when *distance* = 0, other stage variables are unchanged.

Fig. 1: Signal flow graphs of NTT operations, non-faulted and faulted.

$A \cdot x \equiv s \pmod{q}$ where j denotes the position of the element in the NTT output. Every output element $\hat{\mathbf{p}}[j]$ is a function f_j of all the elements of the input sequence $\mathbf{p}$. Due to the very regular structure of the NTT operation and the use of the radix-2 butterfly operation, at each stage the number of input elements on which every output element depends on, uniformly increases by a factor of 2. We refer to Figure 1(a) for a visual representation when $n = 8$. Therefore, after $\log(n)$ stages, every element of the output sequence depends on all the elements of the input sequence. This is referred to as the *diffusion* property of the NTT operation. The primary goal of the attacks and techniques we present is to limit as much as possible such diffusion in the NTT operation. We will see later in the paper that preventing diffusion through fault injections will enable the attacker to break down a faulty LWE instance into many smaller easily solvable systems of equations. Since we are only interested in the diffusion property of the NTT operation, in Figure 1(a) we only highlight the combination of elements at different stages using solid lines. Solid rectangular boxes highlight the various stages of the NTT operation.

In order to better understand how diffusion is achieved, we refer to an open source software implementation of the NTT polynomial multiplier for `AVR` 8-bit micro-controllers by Pöppelmann et al. [29][1]. Refer to Algorithm 1 for the pseudo code of the corresponding NTT transformation.

Algorithm 1 reveals that computation in each stage of the NTT operation depends on three variables. It is important to note how *pairsInGroup*, *distance* and *numOfGroups* are separate variables in this particular implementation but might not be in a different implementation, for instance the ones used by many candidates in the NIST competition. We nonetheless develop our analysis on Algorithm 1 because it is generic enough to the extent that all conclusions we derive will apply to other implementations too, despite technical details differing.

[1] Code available at `https://www.seceng.rub.de/research/projects/pqc/`

Algorithm 1 NTT pseudocode, designed by Pöppelmann *et al.* [29].

```
1: procedure NTT(p, N, q, ψ[·])
2:     pairsInGroup ← N/2
3:     distance ← N/2
4:     numOfGroups ← 1
5:     while numOfGroups < N/2 do
6:         for i = 0 ... numOfGroups do
7:             first ← 2 · i · pairsInGroup
8:             last ← first + pairsInGroup − 1
9:             W ← ψ[numOfGroups + i]
10:            for j = first ... last do
11:                temp ← W · p[j + distance]
12:                p[j + distance] ← p[j] − temp (mod q)
13:                p[j] ← p[j] + temp (mod q)
14:            end for
15:        end for
16:        numOfGroups ← 2 · numOfGroups
17:        distance ← distance/2
18:        pairsInGroup ← pairsInGroup/2
19:    end while
20: end procedure
```

Each stage of the NTT operation computes about $n/2$ butterfly operations. Butterfly operations in each stage are always computed in groups, whose number is determined by the *numOfGroups* variable, while the number of butterfly operations in each group is determined by the *pairsInGroup* variable. The interval between the indices of the input elements to the butterfly operation is determined by the *distance* variable. We refer to these three variables together as *stage variables*, whose values stay constant for a given stage of the NTT operation.

3.1 Faulting the *distance* variable

Among all the stage variables, *distance* directly affects the combination of input operands to any given butterfly operation, so in particular is the main responsible for the diffusion we aim at avoiding. For this reason, our fault attacks focus on tampering with its value.

Consider the computation of a single butterfly, contained in the loop at Line 10 in Algorithm 1

$$\begin{aligned} \mathit{temp} &\leftarrow W \cdot \mathbf{p}[j + \mathit{distance}] \\ \mathbf{p}[j + \mathit{distance}] &\leftarrow \mathbf{p}[j] - \mathit{temp} \pmod q \\ \mathbf{p}[j] &\leftarrow \mathbf{p}[j] + \mathit{temp} \pmod q \end{aligned} \tag{5}$$

We consider faults to ensure a constant value for the *distance* variable throughout all stages of the NTT operation. We identified two possible scenarios of interest that lead to different attack scenarios, that is to say when

1. *distance* is faulted to 0 across all stages. This converts the NTT into a component-wise operation. We will refer to this scenario as *zero setting* for the remaining of this work;
2. *distance* is faulted to a non-zero constant value C for all stages of the NTT operation. We will refer to this scenario as *non-zero setting* for the remaining of this work.

In the non-zero setting, the value C to which *distance* is faulted is of little importance as long as it is within the bound n. From Equation 5 it should be clear that $j + \textit{distance} = j + C$ should lie in the range $[0, \ldots, n-1]$ for the computation to make sense. From now on we assume this is the case and further comment on this point in Section 7. For figures and examples, we will then resort to $C = 1$.

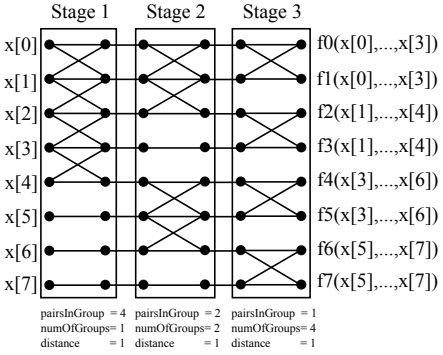
3.1.1 Zero setting. Faulting the *distance* variable to 0 will completely prevent diffusion in the NTT operation. From Equation 5 we can see that both positions affected by one butterfly come from the same index j. Also, both the computed outputs are also written back to the same index j. Thus, we can represent every element of an equivalently faulted NTT operation following the notation used in Equation 4 as follows:

$$\hat{\mathbf{p}}^*[j] = K_j \cdot \mathbf{p}[j] \tag{6}$$

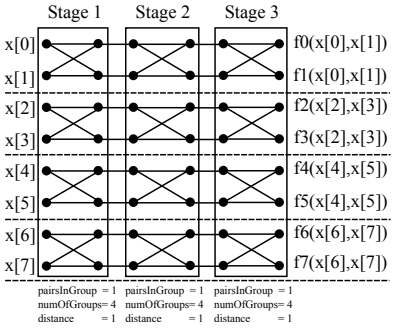
where K_j is a constant value that depends on the structure of the faulted NTT operation, which varies with the index $j \in \{0, n-1\}$, and it is merely a deterministic function of the twiddle factors. Refer Figure 1(b) for a signal flow graph of the corresponding faulted NTT operation. We can clearly see that there is no interaction between any input elements, thus reducing the NTT operation to a component-wise transformation.

3.1.2 Non-zero setting. Faulting the *distance* variable to a non-zero constant C for all stages will lead to different effects depending on whether or not the stage variables are independent of each other. As we mentioned earlier, they are independent for Algorithm 1 but they are dependent for many NIST submissions. Given how important both versions are, we consider them both in our analysis. Refer to Figure 2(a) for the signal flow graph for an NTT operation of Algorithm 1 whose $\textit{distance} = 1$ throughout all the stages while the other stage variables are the same. In this case, we can see that the diffusion is only reduced by half with most of the output elements depend on half the number of input elements. This will still yield a large system of equations for the attacker which is computationally infeasible to be solved for typical parameters. This is the only setting where our analysis does not lead to any concrete attack.

However, for NTT implementations with all the stage variables being derived from a single variable, we can perform targeted faults in such a way to make sure that all stage variables are equivalently affected. This will ensure that operations of a specific stage (depending on the fault injected) are repeated over all the $\log(n)$ stages of the NTT operation. Refer to Figure 2(b) for the signal flow



(a) Faulted NTT when *distance* = 1, stage variables are unchanged.



(b) Faulted NTT when *distance* = 1, with dependent stage variables.

Fig. 2: Signal flow graphs of NTT operations, with independent and dependent *stage-variables*.

graph of the latter type of faulted NTT operation. Thus, we can see that all the output elements can be paired into $n/2$ pairs. One such pair of elements of the corresponding faulty NTT output can be represented as follows:

$$\begin{aligned} \hat{\mathbf{p}}^*[j] &= f_j(\mathbf{p}[j], \mathbf{p}[j+C]) \\ \hat{\mathbf{p}}^*[j+C] &= f_{j+C}(\mathbf{p}[j], \mathbf{p}[j+C]) \ . \end{aligned} \tag{7}$$

In contrast to the zero setting, diffusion in not completely prevented. However, we limit the diffusion of the NTT operation such that each output element depends on only two input elements.

3.2 Effect of faulting the NTT operation

Analysis of lattice-based schemes that utilize the NTT operation reveals that once the NTT transform of a particular polynomial $\hat{\mathbf{s}} = \text{NTT}(\mathbf{s})$ has been computed, all further computations involving $\mathbf{s}$ are performed over $\hat{\mathbf{s}}$ in the NTT domain. Thus, the original polynomial $\mathbf{s}$ can be considered to be discarded by the scheme. This *memory-loss* property of the lattice-based schemes makes it impossible to know if the NTT outputs are valid or not. However, if the NTT transform of $\mathbf{s}$ is faulted to yield $\hat{\mathbf{s}}^*$, the corresponding INTT transform of the faulted NTT output will yield a modified secret polynomial $\mathbf{s}^*$. Thus, this $\mathbf{s}^*$ is used further in the scheme, thus effectively modifying the secret polynomial.

An important caveat present here is that the modified secret $\mathbf{s}^*$, will not satisfy the bounds of the original secret, with very high probability. While the original secret has small coefficients, $\mathbf{s}^*$ is uniformly random in $\mathbb{Z}_q$. This will have profound implications on the corresponding LWE instances which will be discussed in the following sections.

4 Exploiting the fault vulnerability in an LWE instance

In this section, we show how the faults and scenarios described in Section 3 affect the NTT operations to generate faulty LWE instances and the repercussions they have on the security and correctness of the scheme. Let us start by describing a crucial distinction in how LWE instances are generated. Depending on the schemes being considered, the final LWE sample can be either returned in normal domain (e.g. in `KYBER` and `DILITHIUM`) or in NTT domain (e.g. `NEWHOPE`). In the former case, the secret polynomial $\mathbf{s}$ is generated, the NTT is applied and the final LWE sample is computed according to

$$\mathbf{b} = \text{INTT}(\hat{\mathbf{a}} \circ \hat{\mathbf{s}}) + \mathbf{e} \ . \tag{8}$$

In the latter case, instead, the final INTT is not applied and the instance is kept in the NTT domain as follows

$$\hat{\mathbf{b}} = \hat{\mathbf{a}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}} \ . \tag{9}$$

In the present section we analyse both variants.

4.1 Zero setting

Recall that in this case we assume an adversary is able to fault the *distance* variable, in Algorithm 1, to 0. According to Equation 6 from Section 3, we have that each output coefficient of the faulted LWE instance can be represented as

$$\hat{\mathbf{b}}^*[j] = \hat{\mathbf{a}}[j] \cdot K_j \cdot \mathbf{s}[j] + K_j \cdot \mathbf{e}[j]$$

for every $j \in [0 \dots q-1]$. As before, the constants K_j can easily derived from the structure of the faulted NTT operation. Each such equation has 2 unknowns and thus typically have infinitely many solutions. However, it is important to note that both the unknowns $\mathbf{s}[j]$ and $\mathbf{e}[j]$ are sampled from a very narrow distribution with range $[-\eta, \eta]$, we can only have upto $(2\eta+1)^2$ possible number of inputs. But since the number of possible outputs for $\hat{\mathbf{b}}[j]^*$ is q, for typical parameters used in the considered schemes, $q \gg (2\eta+1)^2$. Thus, all the n equations corresponding to each index can be independently solved to arrive at a unique solution or a very few number of solutions for $\mathbf{s}$ with very high probability.

The scenario where the LWE sample is generated in the normal domain is very similar. The equation in this case is

$$\mathbf{b}^*[j] = K_j \cdot L_j \cdot \mathbf{a}[j] \cdot \mathbf{s}[j] + \mathbf{e}[j] \ .$$

The constants K_j and L_j are introduced due to the faulted NTT and INTT transforms, respectively, and can again be easily computed by an adversary. The same conclusion on the number of solutions applies too. We defer the analysis of the above faulted values till the end of the current section (Section 4.3).

4.2 Non-zero setting

In this setting an adversary manages to fault the stage variable *distance* to a non-zero constant C, thus ensuring that each output element only depends on two input elements. We apply Equation 7 to the LWE samples we specified at the beginning of the current section. In NTT domain, this translates to

$$\begin{aligned}\hat{\mathbf{b}}^*[j] &= \mathbf{a}[j] \cdot f_j(\mathbf{s}[j], \mathbf{s}[j+C]) + f_j(\mathbf{e}[j], \mathbf{e}[j+C]) \\ \hat{\mathbf{b}}^*[j+C] &= \mathbf{a}[j+C] \cdot f_{j+C}(\mathbf{s}[j], \mathbf{s}[j+C]) + f_{j+C}(\mathbf{e}[j], \mathbf{e}[j+C]) \ .\end{aligned}$$

Thus, we have a system of 2 linear equations with 4 unknowns, and solving $n/2$ such independent systems will result in retrieval of the entire secret polynomial $\mathbf{s}$. Since the number of possible values of the outputs for one such system, q^2, is much more than the number of possible inputs, $(2\eta+1)^4$, for typical parameters we will be able to get unique solutions for the four coefficients (two each for $\mathbf{s}$ and $\mathbf{e}$) and thus for the complete polynomial $\mathbf{s}$, with very high probability (as in the zero setting). In Section 4.3, we will show that it is in fact more likely to arrive at a unique solution when $\mathit{distance} = C$ rather than when $\mathit{distance} = 0$, albeit at the cost of an higher attack complexity. Unsurprisingly, the case where $\mathbf{b}$ is converted back to normal domain is extremely similar, but we report the equations for completeness:

$$\begin{aligned}\mathbf{b}^*[j] &= g_j(\mathbf{a}[j] \cdot f_j(\mathbf{s}[j], \mathbf{s}[j+C]), \mathbf{a}[j+C] \cdot f_{j+C}(\mathbf{s}[j], \mathbf{s}[j+C])) + \mathbf{e}[j] \\ \mathbf{b}^*[j+C] &= g_{j+C}(\mathbf{a}[j+C] \cdot f_{j+C}(\mathbf{s}[j], \mathbf{s}[j+C]), \mathbf{a}[j] \cdot f_j(\mathbf{s}[j], \mathbf{s}[j+C])) + \mathbf{e}[j+C]\end{aligned}$$

where this time the functions g_j and g_{j+C} can be computed from the INTT structure.

4.3 Analysis of the faulty LWE sample

Thanks to our analysis in Section 3, we derived generic equations stemming from faulting the *distance* variable to either zero (Equation 6) or to a non-zero constant C (Equations 7). In the current section up until now, we applied these equations to LWE samples, both in the setting where they are returned in normal domain and in NTT domain. We specified how the system of equations we derive are underdetermined, thus could potentially have infinitely many solutions. We now present a probabilistic argument based on the solvability of the ISIS problem which shows the reason why the above systems can not only be solved exactly, but also why it is highly unlikely that other solutions other than the correct secret/error exist.

We note that this is the only argument which slightly differs if one considers Ring-LWE and Module-LWE. For this reason we introduce a further parameter r, which indicates the dimension of the secret matrix whose entries are polynomials in $\mathcal{R}$. The case $r = 1$ corresponds to the Ring-LWE problem, while $r > 1$ corresponds to Module-LWE.

In this case, a similar analysis applies to both the zero setting (i.e. $\mathit{distance} = 0$) and the non-zero setting (i.e. $\mathit{distance} = C$). For this reason we focus on the

former, and only at the end comment on the latter. With reference to Equation 1 and Algorithm 1, at stage i of the computation, a butterfly reads the elements at $\mathbf{p}[j]$ and $\mathbf{p}[j + \mathit{distance}]$ and returns $\mathbf{p}[j] = \mathbf{p}[j] + \omega^{i \cdot j}\mathbf{p}[j + \mathit{distance}]$ and $\mathbf{p}[i + \mathit{distance}] = \mathbf{p}[i] - \omega^{i \cdot j}\mathbf{p}[i + \mathit{distance}]$. Hence, if $\mathit{distance} = 0$, we end up with a situation where $\mathbf{p}[i] = \mathbf{p}[i] + \omega^{i \cdot j}\mathbf{p}[i]$ or $\mathbf{p}[i] - \omega^{i \cdot j}\mathbf{p}[i]$, depending on the implementation details, while some elements are never accessed or updated. In general, after stage i, $\mathbf{p}[j] = C_{ij} \cdot \mathbf{p}[j]$ where C_{ij} is a constant value that can be easily computed based on the implementation details.

We look at the effect of this fault on the following sample, where all entries are polynomials in $\mathcal{R}$

$$\begin{bmatrix} \mathbf{a}_1^1 & \mathbf{a}_2^1 & \mathbf{a}_3^1 & \cdots & \mathbf{a}_r^1 \\ \mathbf{a}_1^2 & \mathbf{a}_2^2 & \mathbf{a}_3^2 & \cdots & \mathbf{a}_r^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_1^r & \mathbf{a}_2^r & \mathbf{a}_3^r & \cdots & \mathbf{a}_r^r \end{bmatrix} \cdot \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \\ \mathbf{s}_r \end{bmatrix} + \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \vdots \\ \mathbf{e}_r \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_r \end{bmatrix} . \tag{10}$$

Depending on whether the final output is required to be in normal or in NTT domain, the transformation of the above is computed as

$$\mathbf{b}_i = \mathrm{INTT}(\sum_{j=1}^{r} \hat{\mathbf{a}}_j^i \circ \hat{\mathbf{s}}_j) + \mathbf{e}_i \tag{11}$$

or

$$\hat{\mathbf{b}}_i = \sum_{j=1}^{r} \hat{\mathbf{a}}_j^i \circ \hat{\mathbf{s}}_j + \hat{\mathbf{e}}_i \ . \tag{12}$$

We focus on the latter, and everything follows analogously. By considering the faulted version, we get:

$$\hat{\mathbf{b}}_i^* = \sum_{j=1}^{r} \hat{\mathbf{a}}_j^i \circ \hat{\mathbf{s}}_j^* + \hat{\mathbf{e}}_i^* \tag{13}$$

which can be decomposed into n independent systems of linear equations, each system i being

$$\hat{\mathbf{b}}_i^*[j] = \sum_{k=1}^{r} \mathbf{a}_k^i[j] \cdot c_k \mathbf{s}_k[j] + c_i \mathbf{e}_i[j]. \tag{14}$$

Equation 14 is a succinct description of n systems of linear equations, each consists of $2r$ variables and r equations. Moreover, since the variables are the secret and error coefficients, each of these systems is an $(2r, r, q, \mathcal{B})$-ISIS instance. The complexity of solving each system is $\mathcal{O}(2^r)$ and the density $\delta = |\mathcal{B}|^{2r}/q^r$. If $|B| \ll q$, then δ is close to zero, which means that there is a very low probability of finding a solution. Since at least one solution exists (the secret/error vector), the δ indicates that finding another wrong solution is highly improbable, which means that the number of possible secrets generated by the attack is close to 1. Finally, since n systems exist, the overall attack complexity is $\mathcal{O}(n \cdot 2^r)$. This

result shows a trade-off between the attack complexity and the number of the resulting potential secrets (which is related to the density of the ISIS instance). By varying r, we get the minimum attack complexity for $r = 0$, which is the ring LWE problem, but it also has the maximum density. On the other hand, by increasing r, the density approaches zero, giving a unique solution for almost every instance, but the attack complexity increases exponentially, with the worst case when $r = n$ (General LWE). This goes along with the fact that for General LWE, NTT is a trivial operation and never used.

Non-zero setting. In the case where *distance* is a non-zero constant, the faulty NTT operation acts on pairs of points instead of single points. Consequently, the previous analysis applies, except that only $n/2$ systems are generated and each has $4r$ variables. By increasing the number of variables, the density decreases compared to the case where $distance = 0$, and the attack complexity is squared. The overall complexity and density are $\mathcal{O}(n \cdot 2^{2r})$ and $|\mathcal{B}|^{4r}/q^{2r}$, respectively.

4.4 Different Parameter Sets

Table 1 includes the density of the ISIS instances and an estimation of the attack complexity for the two fault models considered, for different Module-LWE based schemes. All the schemes follow directly from the analysis provided in last section, except for `DILITHIUM`, since the matrix $\bar{\mathbf{a}}$ is not a square matrix. In this case, the resulting ISIS instances are of the form $(k + l, k, q, \mathcal{B})$-ISIS, instead of $(2r, r, q, \mathcal{B})$-ISIS for the rest. But, it is important to note that sometimes LWE instances are not available as it is to the attacker. For example, in the case of the `KYBER` KEM scheme, an additional error is introduced due to the Compress (Decompress) function which has a maximum value of $\lceil \frac{q}{2^{d+1}} \rfloor$ where q is the modulus and d is the number of rounded bits. In the case of `DILITHIUM`, only d MSB bits of each coefficient of the LWE instance are published as the public key to the attacker which introduced an additional error of 2^d to every coefficient of the LWE instance. Numbers in Table 1 have been calculated taking into account the above mentioned additional errors introduced into the LWE instance.

From the table we can see that the maximum brute force complexity is for `NEWHOPE512/1024`, as it has the maximum δ. However, the maximum δ is around 2.4×10^{-2}, which is relatively small enough to ensure a practical attack. We can also see that the estimated attack complexity (AC) ranges between $2^{13.1}$ and $2^{38.7}$, which is practical on a normal computer.

4.4.1 Effect of faulting the LWE sample We would like to refer to the above mentioned attacks which *deconstruct* the LWE instance into smaller easily solvable linear system of equations as *deconstruction* attacks throughout the paper. Irrespective of whether the LWE instance is present in the normal domain or in the NTT domain, such faulted LWE instance when used further in the scheme will violate the correctness of any scheme they are used in. For an LWE

Table 1: Attack complexity (AC) and density (δ) of the ISIS instances for $distance = 0$ and $distance = C$ used for all parameter sets for KYBER, NEWHOPE, and DILITHIUM.

Scheme	Parameters				$distance = 0$		$distance = C$	
	n	**r**	**q**	$\lvert\mathcal{B}\rvert$	δ	AC	δ	AC
KYBER512	256	2	7681	11	4.6×10^{-4}	$2^{14.9}$	2.1×10^{-7}	$2^{20.8}$
KYBER768	256	3	7681	9	3.5×10^{-6}	$2^{17.5}$	1.2×10^{-11}	2^{26}
KYBER1024	256	4	7681	7	1×10^{-8}	$2^{19.2}$	1.0×10^{-16}	$2^{29.5}$
NEWHOPE512	512	1	12289	17	2.4×10^{-2}	$2^{13.1}$	5.5×10^{-4}	$2^{16.2}$
NEWHOPE1024	1024	1	12289	17	2.4×10^{-2}	$2^{14.1}$	5.5×10^{-4}	$2^{17.2}$
DILITHIUM-I	256	$(3, 2)$	8380417	15	1.6×10^{-6}	$2^{15.8}$	2.8×10^{-12}	$2^{22.6}$
DILITHIUM-II	256	$(4, 3)$	8380417	13	3.2×10^{-8}	$2^{19.1}$	1.0×10^{-15}	$2^{29.2}$
DILITHIUM-III	256	$(5, 4)$	8380417	11	4.1×10^{-10}	$2^{21.8}$	1.7×10^{-19}	$2^{34.7}$
DILITHIUM-IV	256	$(6, 5)$	8380417	9	3.3×10^{-12}	$2^{23.8}$	1.0×10^{-23}	$2^{38.7}$

instance in the NTT domain, we fault the NTT transforms of both the secret and error components and thus the modified error component due to the injected fault is no longer small. LWE instances with such large errors which are uniformly distributed will certainly result in very high failure probability in any scheme they are used in.

Referring to Eqn.8 for an LWE instance generated in the normal domain, in order to retrieve the secret from such an instance using the afore mentioned deconstruction attacks, it is required to fault both the NTT transform of $\mathbf{s}$ and the INTT transform of $\mathbf{a} \circ \text{NTT}(\mathbf{s})$. Faulting the NTT transform will lead to use of a modified polynomial $\mathbf{s}^*$ throughout the scheme, while further faulting the INTT transform leads to creation of the LWE instance of scheme corresponding to another secret polynomial $\mathbf{s}^{**}$. Such faulted LWE instances will again violate the correctness of any scheme they are used in.

In both the cases, we can see that such faulted LWE samples can no longer be used in the schemes. So, a natural question arises as to what would the two involved parties do further to either perform a successful key exchange in case of a KEM scheme or a successful authentication in case of a signature scheme. In such a case, it is very much possible that the faulted victim, in a bid to achieve efficiency might attempt to regenerate the LWE instance with the same secret and error inputs that were just retrieved by the attacker. In such a case, an attacker can easily check if a re-use has ocurred and if such a re-use is detected, it translates to secret recovery from a valid LWE instance.

4.5 Note of Caution

We acknowledge that NIST requires the procedures present in all post-quantum cryptographic schemes to be used as cryptographic service calls (i.e) a cryptographic APIs (Application Programming Interface). Thus, if a protocol just calls these procedures as APIs, it is not reasonable for the attacker to expect re-use of

randomness as the sampling operations are present inside the APIs, which get trigerred for every fresh run of the procedure.

But, it is not uncommon to come across implementations that do not adhere to standards or that use certain adhoc techniques to improve their efficiency or performance which might open doors to unknown security vulnerabilities. For example, we know that the Elliptic Curve Digital Signature Algorithm (ECDSA) requires each message to be signed with a unique nonce, but overlooking this simple requirement resulted in a *reuse* scenario leading to the now infamous attack on the PlayStation3 console [33]. Partial reuse of the nonce in elliptic curve signatures is also known to be attackable based on the work of Ambrose et al. [3]. Along the same lines, Adrian et al. [1] reported the famous "LogJam" attack in 2015 on TLS connections wherein one of the main vulnerabilities stemmed from the fact of reuse of operating group across multiple websites and HTTP servers which allowed a single massive precomputation step to amortize the attack over multiple entities. We cite the above examples just to show that overriding of certain known security measures has indeed resulted in practical attacks in the past and thus implementors have to ensure reuse of randomness under any circumstance and that the APIs are called as is from the protocol level to protect against our deconstruction attacks.

5 Additional Vulnerabilities due to use of GS butterfly

The deconstruction attacks discussed previously apply to both NTT transforms that use both the CT and the GS butterfly. In this part, we highlight an additional attack that applies only to the GS butterfly, which arises from its special structure. For this attack to work, we try to create a zero setting i.e. the *distance* variable is faulted to zero. The equivalent of the for loop at Line 10 in Algorithm 1 for a GS butterfly when *distance* is set to zero is as follows:

$$\begin{aligned} temp &\leftarrow \mathbf{p}[j] \\ temp1 &\leftarrow temp + \mathbf{p}[j] \\ temp2 &\leftarrow \omega \cdot (temp - \mathbf{p}[j]) = 0 \\ \mathbf{p}[j] &\leftarrow temp1 \\ \mathbf{p}[j] &\leftarrow temp2 \end{aligned} \tag{15}$$

An equivalently faulted NTT operation can be represented using the signal flow graph in Figure 1(b). If the result of the subtraction operation ($temp2 = 0$) is written after $temp1$, then all the elements in the final NTT output will be 0 except the last element, since no operation is done on the last element. This leaves an attacker with only $2\eta + 1$ possibilities to guess for the last element of the faulty NTT output, while other positions are deterministically zero.

In contrast to what we have presented so far, for this attack to go through, the attacker only has to fault the NTT operation over the secret and thereby due to the zeroing of the elements, creates a modified secret whose NTT output has very low entropy from which he can guess the corresponding modified secret.

Though the modified secrets could be large, valid LWE instances can be created even with large secrets and thus the faulted LWE sample still does not violate the correctness of any scheme it is used in.

This, unlike the previous style of attacks neither requires the victim to reuse randomness in order to result in practical attacks nor work by deconstructing the LWE instance into smaller instances. The zeroing effect upon faults in an GS operation allows the attacker to fault the secret to an easily guessable value with very low entropy. We would like to refer to this attack possible due to the use of GS butterfly as the *zeroing* attack. We acknowledge that such attacks are very implementation specific but the possibility of such attacks must be examined. Thus, in the current setting, we show that the NTT using the GS butterfly NTT has additional vulnerabilities, making CT-butterfly a preferred choice for the NTT operation.

6 Applying the proposed fault analysis on NIST PQC candidates

In this section, we analyse different implementations of the NTT operation used in reference code of the `KYBER` [35], `DILITHIUM` [22], and `NEWHOPE` [27] schemes submitted to the NIST standardization process [24].

We show how the different implementations can be targeted through precise faults to re-create the faulted scenarios ($distance = 0$ and $distance = C$) analysed in the previous section. We find that multiple LWE instances that are generated in the schemes can be targeted using both the deconstruction and zeroing attacks. We also thus analyze various scenarios and situations where it can lead to practical key recovery and message recovery attacks.

Based on our analysis of various implementations of the NTT operation, we can classify the identified fault vulnerabilities in the NTT operation into two categories listed in Table 2.

6.1 Reference Implementation of NTT operation in `NEWHOPE`

The code snippet in Figure 4 shows the NTT^{GS} operation used in the reference implementation of `NEWHOPE`, which is utilized for both the NTT and INTT operations. The *distance* variable is calculated once per stage in Line 5 in even stage (resp. Line 15 in odd stage) using shift operations. It is also possible to skip

Table 2: Types of Fault vulnerabilities in the NTT operation

Fault Vulnerability	**Description**
`FAULT_DIST_CALC:`	Fault the operation calculating the *distance* variable
`FAULT_DIST_UTIL:`	Fault the operation utilizing the *distance* variable

the update of the *distance* value during every stage of the NTT operation, which will ensure a constant value of 1 for the *distance* value. This amounts to $\log n$ instruction skip faults per NTT operation ($distance = C$ using `FAULT_DIST_CALC`). Refer Figure 2(b) shows the signal flow graph of the correspondingly faulted NTT operation.

It is also possible to skip the instruction that utilizes this *distance* variable every butterfly operation. The *distance* variable is used to calculate the address of the second operand of the butterfly operation. This addition instruction which calculates $j + distance$ in Line 11 in even stage (resp. Line 21 in odd stage) can be skipped, which indirectly ensures that $distance = 0$. This has to be repeated every butterfly operation, thus amounting to $(n/2) \log n$ instruction skip faults per NTT operation. Figure 1(b) shows the signal flow graph of the correspondingly faulted NTT operation.

6.1.1 Performing message recovery and key recovery attacks Algorithm 2 shows the key generation and encryption procedures of the `NEWHOPE.CPA.PKE` encryption scheme, which is an integral component of the CCA secure `NEWHOPE` KEM scheme. The public-key is generated as an LWE instance in the NTT domain in Step 11 of the key generation procedure (`NEWHOPE.CPA.PKE.GEN`). Since the NTT^{GS} operation is used, the zeroing attack ($distance = 0$) directly applies here from which the low entropy secret key can be easily guessed from the public key. This only requires faulting the NTT operation of the secret $\mathbf{s}$ (Step 8). But faulting the NTT operations of both $\mathbf{s}$ and $\mathbf{e}$ (Steps 8 and 10, respectively) will create an easily solvable LWE instance from which an attacker can retrieve $\mathbf{s}$ using the deconstruction attack. But, the deconstruction attack requires the victim to re-run the key generation procedure with the same randomness that was used for the faulted run of the protocol in order to result in a valid key recovery attack.

Two LWE instances ($\hat{\mathbf{u}}$ in Step 8 and $\acute{\mathbf{v}}$ in Step 10) are created during the encryption procedure (`NEWHOPE.CPA.PKE.ENC`). Faulting only the NTT operation over the secret $\acute{\mathbf{s}}$ (Step 7) such that $distance = 0$ will result in direct retrieval of $\acute{\mathbf{s}}$ from the faulty LWE instance in Step 8 through the zeroing attack. Alternatively, the attacker can also fault the NTT transforms of both $\acute{\mathbf{s}}$ (Step 7) and $\acute{\mathbf{e}}$ (Step 8) to retrieve $\acute{\mathbf{s}}$ from $\hat{\mathbf{u}}$ through the deconstruction attack. Using the knowledge of $\acute{\mathbf{s}}$, one can easily recover the corresponding message, encoded in the polynomial $\acute{\mathbf{v}}$ as:

$$\mathbf{v} = \acute{\mathbf{v}} - \text{INTT}(\hat{\mathbf{b}} * \hat{\acute{\mathbf{s}}}).$$
$$\mu = \text{Decode}(\mathbf{v})$$

Here again, the zeroing attack leads to valid message recovery attack, while such an attack is possible using the deconstruction attack only when the same randomness is re-used for the second run of the encryption procedure. Else another new secret and error polynomial pair is used for the second run of the encryption procedure.

6.2 Reference Implementation of NTT operation in KYBER

The NTT^{CT} and INTT^{GS} algorithms are used in the reference implementation of KYBER and since the stage variables are handled in the same way in both the algorithms, we perform the analysis over code snippet of NTT^{CT} in Figure 5 which also directly applies to the implementation of the INTT^{GS} algorithm.

The variable $(1 \ll \textit{level})$ is used as the *stage-variable* which is calculated (Line 6) from the *level* variable that is used as the counter for the number of stages of the NTT operation (Line 5). The attacker can skip the calculation of the stage variable (shift operation in Line 6) to ensure a constant value of 1 for distance throughout the NTT operation (FAULT_DIST_CALC). This is achievable through $\log(n)$ instruction skip faults. Figure 2(b) shows the corresponding signal flow graph of the faulted NTT operation.

The $(1 \ll \textit{level})$ variable is used for calculating the address of the second input to the butterfly operation (Line 9). Here again, the attacker can skip this addition instruction $(j + (1 \ll \textit{level}))$ to create a scenario of $\textit{distance} = 0$ (FAULT_DIST_UTIL), which requires around $(n/2) \log n$ faults per NTT transform. Figure 1(b) shows the corresponding signal flow graph of the faulted NTT operation.

6.2.1 Performing message recovery and key recovery attacks Algorithm 3 shows the key generation and encryption procedures of the KYBER.CPA.PKE encryption scheme. Zeroing attacks are not possible due to the use of CT-butterfly for the forward NTT operation. The NTT and INTT operations which are performed in Steps 18 and 19 of the key generation procedure (KYBER.CPAPKE.GEN) are required to be faulted to generate a faulty LWE instance ($\hat{\mathbf{t}}$). Additional error is added to the LWE instance ($\mathbf{t}$) using the Compress function (Step 20) and is published as the public key pk. The attacker can perform the deconstruction attack over pk to recover the secret key $\mathbf{s}$. Though the generated faulty LWE instance might violate the correctness of the scheme, re-use of randomness (Step 2) for the second run of the key generation procedure will result in a valid key recovery attack.

Similarly, faulting NTT and INTT operations in Step 12 and Step 13 of the encryption procedure (KYBER.CPA.PKE.ENC) will create a faulty LWE instance $\hat{\mathbf{u}}$ from which one can easily retrieve the corresponding secret $\mathbf{r}$ through the deconstruction attack. Knowing $\mathbf{r}$, one can easily retrieve the encoded secret message m in $\mathbf{c}_2$ using a similar technique used in the attack over NEWHOPE. Though this faulted encryption procedure will result in a failure, if the same secret r is used in the re-run of the encryption procedure, it will result is a valid message recovery attack.

6.3 Reference implementation of NTT operation in DILITHIUM

The NTT^{CT} and INTT^{GS} operations are used in the reference implementation of DILITHIUM and since the stage variables are handled in the same way in both

the algorithms, we only analyse the code snippet of NTT^{CT} in Figure 6, which also directly applies to the implementation of the INTT^{GS} algorithm.

The implementation style of the NTT operation used in the reference implementation of `DILITHIUM` differs from that used in `NEWHOPE` and `KYBER` in a very subtle manner. The *len* variable is used as both the *stage-variable* as well as the stage counter of the NTT operation (Line 5), while both the stage counter and the *stage-variable* are independent of each other in the case of `NEWHOPE` and `KYBER`. Thus directly faulting the *len* variable also affects the number of stages computed in the NTT operation.

An attacker can alternatively fault *distance* to 0 indirectly by targetting the instruction utilizing the *len* variable (`FAULT_DIST_UTIL`). Skipping the addition operation ($j + len$) (Line 9) that is used to calculate the address of the second operand to the butterfly operation can be utilized to create a scenario similar to when $distance = 0$. This again requires around $(n/2) \log n$ faults per NTT operation. Figure 1(b) shows the signal flow graph of the correspondingly faulted NTT operation.

6.3.1 Performing key recovery attacks Algorithm 4 shows the key generation procedure of the `DILITHIUM` signature scheme. The only LWE instance generated is the public-key and hence it is only possible to perform key recovery attacks. Zeroing attacks cannot be performed since the CT-butterfly is used for the forward NTT transformation. The LWE instance $\mathbf{t}$ is created in Step 12 of the key generation procedure `DILITHIUM.KEYGEN()` but only the d higher order bits of every coefficient is revealed as the public key ($pk = \mathbf{t}_1 = \text{Power2Round}(\mathbf{t})$). By faulting the NTT and INTT operations in Step 12, a deconstruction attack can still be applied over the public key to retrieve the secret key. The density of the corresponding smaller ISIS instances to be solved still have densities low enough to guarantee a unique solution, which is also verified by practical results from Section 7.1. But here again, the correctness of the signature scheme cannot be ensured with a faulty public-key. But, if the valid signer reuses the randomness (Step 2) to generate the valid public-key, then it results in a valid key recovery attack.

Table 3 shows the summary of the fault complexity of our attack on reference implementations of the NTT operation across the recommended parameter sets of the considered schemes. It is important to also understand how the attacker knows that all the injected faults are successful. Since the secret and error components are small, if the faults were not injected as intended, then the retrieved secret and error components from the faulty LWE instances would be large with very high probability. This indirectly and implicitly helps the attacker to examine the validity of all the injected faults together for one run of the procedure.

7 Experimental Results

To validate the proposed fault attacks, we have conducted a series of laser fault injection experiments. Our setup is depicted in Figure 3(a). The core of the setup

Table 3: Fault complexity of our attack on various recommended parameter sets of the proposed schemes.

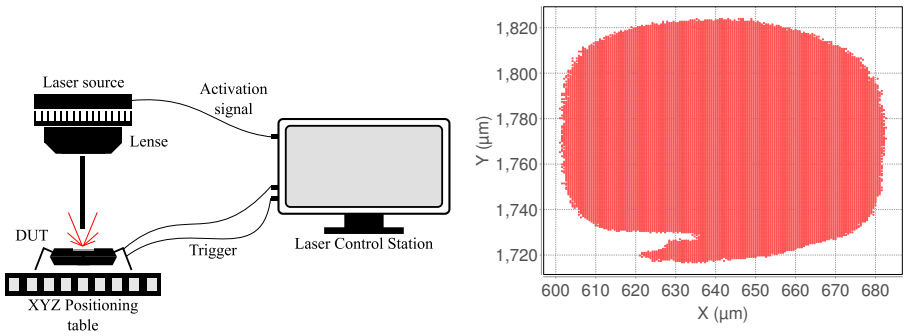
Scheme	Type of Vulnerability	Type of Fault	Type of Attack	*distance*	Faults Per NTT	Total Faults
NEWHOPE512	FAULT_DIST_UTIL	Skip Add	Deconstruct	0	2304	4608
	FAULT_DIST_UTIL	Skip Add	Zeroing	0	2304	2304
	FAULT_DIST_CALC	Skip Shift	Deconstruct	C	9	18
NEWHOPE1024	FAULT_DIST_UTIL	Skip Add	Deconstruct	0	5120	10240
	FAULT_DIST_UTIL	Skip Add	Zeroing	0	5120	5120
	FAULT_DIST_CALC	Skip Shift	Deconstruct	C	10	20
KYBER512	FAULT_DIST_UTIL	Skip Add	Deconstruct	0	1024	4096
	FAULT_DIST_CALC	Skip Shift		C	8	32
KYBER768	FAULT_DIST_UTIL	Skip Add	Deconstruct	0	1024	6144
	FAULT_DIST_CALC	Skip Shift		C	8	48
KYBER1024	FAULT_DIST_UTIL	Skip Add	Deconstruct	0	1024	8192
	FAULT_DIST_CALC	Skip Shift		C	8	64
DILITHIUM-I	FAULT_DIST_UTIL	Skip Add	Deconstruct	0	1024	5120
DILITHIUM-II	FAULT_DIST_UTIL	Skip Add	Deconstruct	0	1024	7168
DILITHIUM-III	FAULT_DIST_UTIL	Skip Add	Deconstruct	0	1024	9216
DILITHIUM-IV	FAULT_DIST_UTIL	Skip Add	Deconstruct	0	1024	11264

is a near-infrared (1064 nm) pulse laser diode with power of 20 W, reduced to 8 W by using 20x objective lens. The spot size after 20x magnification is 15×3.5 μm. The pulse repetition rate is 10 MHz. Device under test (DUT) was mounted on the X-Y positioning table with a step precision of 0.05 μm.

As for the DUT, we used an 8-bit AVR ATmega328P micro-controller, operating at 16 MHz. The whole chip area is 3×3 mm, and after a thorough profiling phase, it was possible to precisely localize the sensitive area of the chip and inject faults with a very high repeatability rate. The timing of the fault injection was triggered by a high voltage signal (5 V), coming from one of the output pins, set to start at the beginning of the encryption.

The first step was profiling, used to determine the vulnerable area of the chip. This area was approximately 75×100 μm large (≈0.083% of the chip area) and is depicted in Figure 3(b). The glitch length varied between 50-150 ns, and the laser power varied between 3-4.5%. Timing of the laser glitch was precisely set to each of the targeted instructions – following the clock frequency, one clock takes 62.5 ns, while the glitch activation can vary with nanosecond precision. Scanning of the chip was done with 150 steps in each direction, resulting to 22,500 experiments. The time required for such scan is ≈ 2 hours. The fault injection resulted in a faulty output from the encryption routine, at different co-ordinates.

After finding the appropriate location and fixing the laser parameters to the optimal ones after the profiling phase, we have conducted series of experiments to check the possible fault models and the repeatability. We have put reference implementations of the target schemes in the device and conducted series of tests to determine the fault model. Our analysis showed that the most favorable



(a) (b)

Fig. 3: (a) Laser fault injection experimental setup (b) Area of the chip vulnerable to laser (full coordinates of the chip are $(0, 3000)$ in each direction).

model was instruction skip – the repeatability was very high and any instruction could be targeted. We have tested following instructions in accordance to this fault model: `ADD` and `LSR`, since skipping them would allow us to do the fault analysis described in the previous sections. Glitch length was set to 150 ns and the laser power was set to 4.5%. For each of the instructions we have conducted 1000 instruction skip experiments, resulting to 100% repeatability. Therefore, we can conclude that after getting the necessary parameters, the attacker can carry out the fault attack with just one laser experiment. It is worth mentioning that in case where several fault injections are necessary during one run of an operation (For eg. NTT transform), it is possible to follow a pre-defined pattern that will turn the laser on and off. While the the first pulse activation after the trigger is $\approx$ 100 ns, the consecutive glitches can be activated with the precision of $\leq$ 60 ns. After deriving the practical fault models from the target device, in the following we compute the attacker's computational complexity for the decomposition attacks under observed faults.

7.1 Key Recovery Analysis for Deconstruction Attacks

Following successful injection of the faults in the `NTT` transform, the attacker can construct a certain number of independent system of equations (ISIS instances) from the faulty LWE sample. These ISIS instances can be independently solved to retrieve all the unknown coefficients of the secret using a divide and conquer strategy. Multiple solutions usually exist for overdefined system of equations, but as seen in Section 4, the probability of arriving at a unique solution for $\mathbf{s}$ depends on the density of the corresponding ISIS instance.

In order to validate these claims, we generated multiple faulty LWE instances using fault simulations, based on the different types of fault vulnerabilities identified in Section 6 and fault model validated on the device. We performed our experiments on an Intel 2.6 GHz Core-i5 (Haswell) processor. Table 4 shows

Table 4: Results for Key recovery analysis from deconstruction attacks based on the identified fault vulnerabilities for parameter sets of various schemes

Scheme	Type of Vulnerability	Type of Fault	Brute-Force Complexity	Time (secs)
NEWHOPE512	FAULT_DIST_UTIL	Skip Add	2557	0.063
	FAULT_DIST_CALC	Skip Shift	1.15	0.790
NEWHOPE1024	FAULT_DIST_UTIL	Skip Add	8240719	0.251
	FAULT_DIST_CALC	Skip Shift	1.44	2.531
KYBER512	FAULT_DIST_UTIL	Skip Add	1.08	8.171
	FAULT_DIST_CALC	Skip Shift	1	716.317
KYBER768	FAULT_DIST_UTIL	Skip Add	1	12.314
	FAULT_DIST_CALC	Skip Shift	1	4.6 hrs
KYBER1024	FAULT_DIST_UTIL	Skip Add	1	17.359
	FAULT_DIST_CALC	Skip Shift	1	44.5 hrs
DILITHIUM-I	FAULT_DIST_UTIL	Skip Add	1	4.807
DILITHIUM-II	FAULT_DIST_UTIL	Skip Add	1	45.429
DILITHIUM-III	FAULT_DIST_UTIL	Skip Add	1	342.396
DILITHIUM-IV	FAULT_DIST_UTIL	Skip Add	1	304.208

the results of average brute force complexity and time required to perform key recovery in each of these cases.

Analysis of results in Table 4 reveals that the brute force complexity and the attacker's computational complexity are inversely proportional to each other. With increasing *diffusion* in the NTT transform, the attacker is more likely to retrieve a unique solution for the secret $\mathbf{s}$, while the computational complexity increases exponentially with increasing number of variables to be brute-forced. Another claim which can be validated from the numbers in the table is that the faulty Module-LWE instances yield smaller brute force complexity compared to their Ring-LWE counterparts. This is due to the fact that density of instances is inherently small and decreases exponentially, while the attacker's computational complexity (time taken) increases exponentially with increasing rank of the module.

7.2 Extension to Other Targets

The proposed fault attacks in the paper require conditions $distance = 0$ or $distance = C$. We experimentally achieved this conditions under a instruction skip model by analysing the public reference implementations of target schemes submitted to NIST competition. While instruction skip is limited to software (or microcontroller) targets, the proposed attack is not. It is noteworthy that other fault models can also result in a equivalent faults. For example, targeted bit flips can be used to force a condition of $distance = 0$ or $distance = C$. Thus the proposed attacks apply to range of devices including FPGA and ASIC. The attacker should adapt equipment capability and injection strategy accordingly.

8 Conclusion

In this paper, we perform a fault vulnerability and fault propogation analysis of the NTT operation, which is heavily used in the context of lattice-based cryptography. We identify two types of attacks, the deconstruction attacks which fault the NTT operations in such a way so as to generate faulty LWE instances which can be decomposed into easily solvable ISIS instances. While these attacks require the reuse of randomness constraint to result in valid key and message recovery attacks, we also identify a special zeroing attack that works on NTT operations using the GS butterfly. The zeroing attack does not require the constraint of reuse of randomness by the victim, thus poses as a stronger attack. We analyse the various implementations of the NTT operation used in reference implementations of the considered NIST PQC candidates and show how these vulnerabilities can be exploited to lead to practical message recovery and key recovery attacks in certain scenarios. We provide simulation results and practical laser fault injection experiments, targeting an AVR microcontroller, in addition to theoretical results, which complete a comprehensive fault analysis of the NTT. In the future, it would be worth extending the proposed attacks to other lattice-based cryptographic schemes and deriving a method to protect the NTT that would make fault injection attacks infeasible.

References

1. Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., et al.: Imperfect forward secrecy: How diffie-hellman fails in practice. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 5–17. ACM (2015)
2. Akleylek, S., Bindel, N., Buchmann, J., Krämer, J., Marson, G.A.: An efficient lattice-based signature scheme with provably secure instantiation. In: International Conference on Cryptology in Africa. pp. 44–60. Springer (2016)
3. Ambrose, C., Bos, J.W., Fay, B., Joye, M., Lochter, M., Murray, B.: Differential attacks on deterministic signatures. In: CryptographersâĂŹ Track at the RSA Conference. pp. 339–353. Springer (2018)
4. Bai, S., Galbraith, S.D., Li, L., Sheffield, D.: Improved combinatorial algorithms for the inhomogeneous short integer solution problem. Tech. rep., IACR Cryptology ePrint Archive, 2014: 593 (2014)
5. Bindel, N., Buchmann, J., Krämer, J.: Lattice-based signature schemes and their sensitivity to fault attacks. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016 Workshop on. pp. 63–77. IEEE (2016)
6. Bindel, N., Krämer, J., Schreiber, J.: Hampering fault attacks against lattice-based signature schemes: countermeasures and their efficiency (special session). In: Proceedings of the Twelfth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion. p. 8. ACM (2017)
7. Blömer, J., d. Silva, R.G., GÃijnther, P., Krämer, J., Seifert, J.P.: A practical second-order fault attack against a real-world pairing implementation. In: 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 123–136 (Sept 2014)

8. Breier, J., Jap, D., Chen, C.N.: Laser profiling for the back-side fault attacks: With a practical laser skip instruction attack on aes. In: Proceedings of the 1st ACM Workshop on Cyber-Physical System Security. pp. 99–103. CPSS '15, ACM, New York, NY, USA (2015)
9. Chen, H., Lauter, K., Stange, K.E.: Attacks on search RLWE. https://www.microsoft.com/en-us/research/publication/attacks-on-search-rlwe/ (2015)
10. Cooley, J.W., Lewis, P.A., Welch, P.D.: Historical notes on the fast fourier transform. Proceedings of the IEEE 55(10), 1675–1677 (1967)
11. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal gaussians. In: Advances in Cryptology–CRYPTO 2013, pp. 40–56. Springer (2013)
12. Elias, Y., Lauter, K.E., Ozman, E., Stange, K.E.: Provably weak instances of Ring-LWE. In: Annual Cryptology Conference. pp. 63–92. Springer (2015)
13. Espitau, T., Fouque, P.A., Gérard, B., Tibouchi, M.: Loop abort faults on lattice-based fiat-shamir & hash'n sign signatures. IACR Cryptology ePrint Archive 2016, 449 (2016)
14. Espitau, T., Fouque, P.A., Gérard, B., Tibouchi, M.: Side-channel attacks on bliss lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1857–1874. ACM (2017)
15. Gentleman, W.M., Sande, G.: Fast fourier transforms: for fun and profit. In: Proceedings of the November 7-10, 1966, fall joint computer conference. pp. 563–578. ACM (1966)
16. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: STOC. pp. 212–219. ACM (1996)
17. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: A signature scheme for embedded systems. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 530–547. Springer (2012)
18. Harty, T., Allcock, D., Ballance, C.J., Guidoni, L., Janacek, H., Linke, N., Stacey, D., Lucas, D.: High-fidelity preparation, gates, memory, and readout of a trapped-ion quantum bit. Physical review letters 113(22), 220501 (2014)
19. Kumar, S.D., Patranabis, S., Breier, J., Mukhopadhyay, D., Bhasin, S., Chattopadhyay, A., Baksi, A.: A practical fault attack on arx-like ciphers with a case study on chacha20. In: 2017 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC, Taipei, Taiwan (2017)
20. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Designs, Codes and Cryptography 75(3), 565–599 (2015)
21. Lyubashevsky, V.: Lattice signatures without trapdoors. In: EUROCRYPT. pp. 738–755 (2012)
22. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehle, D.: Crystals-dilithium. Tech. rep., National Institute of Standards and Technology (2017)
23. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM 60(6), 43 (2013)
24. NIST: Post-quantum crypto project. `http://csrc.nist.gov/groups/ST/post-quantum-crypto/` (2016)
25. Nussbaumer, H.: Fast Fourier transform and convolution algorithms. Springer-Verlag (1980)

26. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical cca2-secure and masked ring-lwe implementation
27. Poppelmann, T., Alkim, E., Avanzi, R., Bos, J., Ducas, L., de la Piedra, A., Schwabe, P., Stebila, D.: Newhope. Tech. rep., National Institute of Standards and Technology (2017)
28. Pöppelmann, T., Ducas, L., Güneysu, T.: Enhanced lattice-based signatures on reconfigurable hardware. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 353–370. Springer (2014)
29. Pöppelmann, T., Oder, T., Güneysu, T.: Speed records for ideal lattice-based cryptography on avr. IACR Cryptology ePrint Archive 2015, 382 (2015)
30. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005. pp. 84–93 (2005)
31. Riviere, L., Najm, Z., Rauzy, P., Danger, J.L., Bringer, J., Sauvage, L.: High precision fault injections on the instruction cache of armv7-m architectures. In: Hardware oriented security and trust (host), 2015 ieee international symposium on. pp. 62–67. IEEE (2015)
32. Roy, S.S., Vercauteren, F., Mentens, N., Chen, D.D., Verbauwhede, I.: Compact ring-LWE cryptoprocessor. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 371–391. Springer (2014)
33. Schmid, M.: Ecdsa-application and implementation failures (2015)
34. Schroeppel, R., Shamir, A.: A t=o(2^n/2), s=o(2^n/4) algorithm for certain np-complete problems. SIAM journal on Computing 10(3), 456–464 (1981)
35. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehle, D.: Crystals-kyber. Tech. rep., National Institute of Standards and Technology (2017)
36. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (Oct 1997)

A Appendices

A.1 Code Snippets of NTT implementations used in reference implementations of NIST PQC candidates

This section lists the code snippets of the NTT transform implementations used in the reference code of NEWHOPE, KYBER and DILITHIUM schemes, submitted to the first round of the NIST standardization process.

A.2 Description of procedures used in various NIST PQC candidates

This section provides details of the key generation and encryption procedures of the KEM schemes NEWHOPE and KYBER and the key generation procedure of the DILITHIUM signature scheme. We only list the afore mentioned procedures of the respective schemes since they contain the target operations which remain the focus of our fault attack.

```
void ntt(uint16_t * a, const uint16_t* omega){
  int i, start, j, jTwiddle, distance;
  uint16_t temp, W;
  for(i=0;i<9;i+=2){
    distance = (1<<i);
    for(start = 0; start < distance;start++){
      jTwiddle = 0;
      for(j=start;j<NEWHOPE_N-1;j+=2*distance){
       W = omega[jTwiddle++];
       temp = a[j];
       a[j] = (temp + a[j+distance]);
       a[j+distance] = montgomery_reduce((W*((uint32_t)temp
                              + 3*NEWHOPE_Q - a[j+distance])));}}
    if(i+1<9){
      distance <<= 1;
      for(start = 0; start < distance;start++){
        jTwiddle = 0;
        for(j=start;j<NEWHOPE_N-1;j+=2*distance){
         W = omega[jTwiddle++];
         temp = a[j];
         a[j] = (temp + a[j+distance]) % NEWHOPE_Q;
         a[j+distance] = montgomery_reduce((W*((uint32_t)temp
                                + 3*NEWHOPE_Q - a[j+distance])));
         }}}}}
```

Fig. 4: Code Snippet of NTT transformation (Both NTT and INTT) as used in reference implementation of `NEWHOPE`

```
void ntt(uint16_t *p){
  int level, start, j, k;
  uint16_t zeta, t;
  k = 1;
  for(level=7; level>= 0; level--){
    for(start=0; start<KYBER_N; start=j+(1<<level)){
      zeta = zetas[k++];
      for(j=start; j<start+(1<<level); ++j){
       t = montgomery_reduce((uint32_t)zeta*p[j+(1<<level)]);
       p[j+(1<<level)] = barrett_reduce(p[j]+4*KYBER_Q-t);
       if(level & 1) /* odd level */
         p[j] = p[j] + t;
       else
         p[j] = barrett_reduce(p[j] + t);}}}}
```

Fig. 5: Code Snippet of NTT transformation used in reference implementation of `KYBER`

```
void ntt(uint32_t p[N]){
  unsigned int len, start, j, k;
  uint32_t zeta, t;
  k = 1;
  for(len = 128; len > 0; len >>= 1){
    for(start = 0; start < N; start = j + len){
      zeta = zetas[k++];
      for(j = start; j < start + len; ++j){
        t = montgomery_reduce((uint64_t)zeta * p[j + len]);
        p[j + len] = p[j] + 2*Q - t;
        p[j] = p[j] + t;}}}}
```

Fig. 6: Code Snippet of NTT transformation as used in reference implementation of DILITHIUM

Algorithm 2 NEWHOPE CPA-PKE scheme

procedure NEWHOPE.CPAPKE.GEN()
  $seed \leftarrow \{0,\ldots,255\}^{32}$
  $z \leftarrow \mathsf{SHAKE256}(64, seed)$
  $publicseed \leftarrow z[0:31]$
  $noiseseed \leftarrow z[32:63]$
  $\mathbf{a} \leftarrow \mathsf{GenA}(publicseed)$
  $\mathbf{s} \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(noiseseed, 0))$
  $\hat{\mathbf{s}} = \mathsf{NTT}(\mathbf{s})$
  $\mathbf{e} \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(noiseseed, 1))$
  $\hat{\mathbf{e}} = \mathsf{NTT}(\mathbf{e})$
  $\hat{\mathbf{b}} = \mathbf{a} * \hat{\mathbf{s}} + \hat{\mathbf{e}}$
  Return $(pk = \mathsf{EncodePK}(\hat{\mathbf{b}}, publicseed), sk = \mathsf{EncodePolynomial}(\mathbf{s}))$
end procedure

procedure NEWHOPE.CPAPKE.ENC($pk \in \{0,\ldots,255\}^{7.n/4+32}, \mu \in \{0,\ldots,255\}^{32}, coin \in \{0,\ldots,255\}^{32}$)
  $\mathbf{b}, \hat{publicseed} \leftarrow \mathsf{DecodePk}(pk)$
  $\mathbf{a} \leftarrow \mathsf{GenA}(publicseed)$
  $\acute{\mathbf{s}} \leftarrow \mathsf{PolyBitRev}(Sample(coin, 0))$
  $\acute{\mathbf{e}} \leftarrow \mathsf{PolyBitRev}(Sample(coin, 1))$
  $\acute{\mathbf{e}} \leftarrow \mathsf{Sample}(coin, 2)$
  $\hat{\acute{\mathbf{s}}} = \mathsf{NTT}(\acute{\mathbf{s}})$
  $\hat{\mathbf{u}} = \mathbf{a} * \hat{\acute{\mathbf{s}}} + \mathsf{NTT}(\acute{\mathbf{e}})$
  $\mathbf{v} = \mathsf{Encode}(\mu)$
  $\acute{\mathbf{v}} = \mathsf{INTT}(\hat{\mathbf{b}} * \hat{\acute{\mathbf{s}}}) + \acute{\mathbf{e}} + \mathbf{v}$
  $\mathbf{h} = \mathsf{Compress}(\acute{\mathbf{v}})$
  Return $c = \mathsf{EncodeC}(\hat{\mathbf{u}}, \mathbf{h})$
end procedure

Algorithm 3 KYBER CPA-PKE scheme

procedure KYBER.CPAPKE.GEN()
  $d \leftarrow \{0,1\}^{256}$
  $(\rho, \sigma) := G(d)$
  $N := 0$
  for i from 0 to $k-1$ **do**
    for j from 0 to $k-1$ **do**
      $\mathbf{a}[i][j] \leftarrow \mathsf{Parse}(\mathsf{XOF}(\rho||j||i))$
    end for
  end for
  for i from 0 to $k-1$ **do**
    $\mathbf{s}[i] \leftarrow \mathsf{CBD}_\eta(\mathsf{PRF}(\sigma, N))$
    $N := N + 1$
  end for
  for i from 0 to $k-1$ **do**
    $\mathbf{e}[i] \leftarrow \mathsf{CBD}_\eta(\mathsf{PRF}(\sigma, N))$
    $N := N + 1$
  end for
  $\hat{\mathbf{s}} \leftarrow \mathsf{NTT}(\mathbf{s})$
  $\mathbf{t} = \mathsf{INTT}(\mathbf{a} * \hat{\mathbf{s}}) + \mathbf{e}$
  $pk := (\mathsf{Encode}_{d_t}(\mathsf{Compress}_q(\mathbf{t}, d_t))||\rho)$
  $sk := \mathsf{Encode}_{13}(\hat{\mathbf{s}}\ mod^+ q)$
  Return (pk, sk)
end procedure

---

procedure KYBER.CPAPKE.ENC($pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}$, $m \in \mathcal{B}^{32}$, $r \in \mathcal{B}^{32}$)
  $N = 0$
  for i from 0 to $k-1$ **do**
    $\mathbf{r}[i] \leftarrow \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$
    $N := N + 1$
  end for
  for i from 0 to $k-1$ **do**
    $\mathbf{e}_1[i] \leftarrow \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$
    $N := N + 1$
  end for
  $\mathbf{e}_2 \leftarrow \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$
  $\hat{\mathbf{r}} = \mathsf{NTT}(\mathbf{r})$
  $\mathbf{u} = \mathsf{INTT}(\mathbf{a}^T * \hat{\mathbf{r}}) + \mathbf{e}_1$
  $\mathbf{v} = \mathsf{INTT}(\mathbf{t}^T * \hat{\mathbf{r}}) + \mathbf{e}_2 + \mathsf{Decode}_1(\mathsf{Decompose}_q(m, 1))$
  $\mathbf{c}_1 = \mathsf{Encode}_{d_u}(\mathsf{Compress}_q(\mathbf{u}, d_u))$
  $\mathbf{c}_2 = \mathsf{Encode}_{d_v}(\mathsf{Compress}_q(\mathbf{v}, d_v))$
  $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$
end procedure

Algorithm 4 DILITHIUM Key Generation

procedure DILITHIUM.KEYGEN()
  $\rho, \rho' \leftarrow \{0,1\}^{256}$
  $K \leftarrow \{0,1\}^{256}$
  $N := 0$
  for i from 0 to $\ell - 1$ **do**
    $\mathbf{s}_1[i] := Sample(PRF(\rho', N))$
    $N := N + 1$
  end for
  for i from 0 to $k - 1$ **do**
    $\mathbf{s}_2[i] := Sample(PRF(\rho', N))$
    $N := N + 1$
  end for
  $\mathbf{A} \sim R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$
  Compute $\mathbf{t} = \mathsf{INTT}(\mathbf{A} * \mathsf{NTT}(\mathbf{s}_1)) + \mathbf{s}_2$
  Compute $\mathbf{t}_1 := \mathsf{Power2Round}_q(\mathbf{t}, d)$
  $tr \in \{0,1\}^{384} := \mathsf{CRH}(\rho || \mathbf{t}_1)$
  Return $pk = (\rho, \mathbf{t}_1), sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
end procedure