A Fault-Resistant NTT by Polynomial Evaluation and Interpolation

Application to Kyber and Dilithium

Sven Bauer[0000−0003−1882−6110], Fabrizio De Santis[0000−0003−3194−826X], Kristjane Koleci[0000−0003−2781−6379], and Anita Aghaie[0000−0003−2470−3408]

Siemens AG, Technology, Munich, Germany
{svenbauer, fabrizio.desantis,
kristjane.koleci, anita.aghaie}@siemens.com

Abstract. In computer arithmetic operations, the Number Theoretic Transform (NTT) plays a significant role in the efficient implementation of cyclic and nega-cyclic convolutions with the application of multiplying large integers and large degree polynomials. Multiplying polynomials is a common operation in lattice-based cryptography. Hence, the NTT is a core component of several lattice-based cryptographic algorithms. Two well-known examples are the key encapsulation mechanism Kyber and the digital signature algorithm Dilithium. In this work, we introduce a novel and efficient method for safeguarding the NTT against fault attacks. This new countermeasure is based on polynomial evaluation and interpolation. We prove its error detection capability, calculate the required additional computational effort, and show how to concretely use it to secure the NTT in Kyber and Dilithium against fault injection attacks. Finally, we provide concrete implementation results of the proposed novel technique on a resource-constrained ARM Cortex-M4 microcontroller, e.g., the technique exhibits a 72% relative overhead, when applied to Dilithium.

Keywords: Lattice-Based Cryptography · Post-Quantum Cryptography · Kyber · Dilithium · NTT · Fault Countermeasures.

1 Introduction

The Number Theoretic Transform (NTT) is a core building block of a number of cryptographic schemes defined over polynomials rings. It plays a central role in various lattice-based cryptographic schemes that rely on the difficulty of certain computational problems in structured lattices.

Both the post-quantum key encapsulation mechanism Kyber [23] and the post-quantum digital signature scheme Dilithium [17] make use of the NTT to efficiently compute polynomial multiplication. Both algorithms have recently been selected by the US National Institute of Standards and Technology (NIST) for final standardization [19]. The initial draft standards have recently been published for public comment and review as FIPS 203 [2] and FIPS 204 [3] for Kyber and Dilithium, respectively.

Kyber and Dilithium were designed with NTT-friendly parameters, in order to allow for an efficient implementation of polynomial multiplication using the NTT.

While the NTT provides significant benefits in terms of speed and memory, it is also an attractive target for side-channel and fault attacks [22, 15]. While there have been several studies focusing on protecting the NTT against side-channel attacks [21, 18, 10, 6, 8], there has been comparatively little research conducted on the fault resistance of the NTT itself [22, 6].

Related Work In the context of lattice-based cryptography, a number of fault attacks have been presented. For instance, various fault attacks against BLISS, ring-TESLA, and the GLP-scheme have been reported in [7]. Differential fault attacks against deterministic variants of Dilithium and Falcon have been presented in [9] and [4]. Fault attacks against signature verification in Dilithium and Falcon have been considered in [20, 22, 5].

In [14], a chosen-ciphertext fault attack against Kyber is introduced where the fault can be injected during almost the entire decapsulation or at more specific locations during re-encryption. This proposed approach involves manipulating the ciphertext and correcting it by fault injection to obtain inequalities and recover the secret key using belief propagation. It has been demonstrated that this method can bypass several countermeasures such as straightforward shuffling and boolean masking methods [14]. In [24], single instruction skip fault injections during the decapsulation are considered for various KEM algorithms such as Kyber. More precisely, the attack approach involves exploiting the Fujisaki-Okamoto (FO) transform used in Kyber. The attacker implements a skipping-the-equality-test attack by carefully injecting faults during the decapsulation process. This fault injection causes the algorithm to skip a critical equality test between the original and re-encrypted ciphertexts, consequently bypassing this security check. It has been shown that this method can be effective in compromising the security of Kyber implementations. In addition, the attack has been improved in [11] considering single bit flips.

Regarding attacks on the NTT itself, several studies have demonstrated that the NTT operation is susceptible to fault attacks, as outlined below. In [12] the attack presented in [14] has been improved to include binomial sampling and NTT butterflies and by relaxing the fault model to include random faults and instruction skips. This work also shows that the countermeasure proposed in [14] is not effective against the improved attacks. In [22], a number of fault attacks against the NTT are demonstrated, hence highlighting the criticality of safeguarding this operation against fault attacks. In more detail, the proposed attack approach on NTT in [22] involves manipulating the twiddle factors in NTT implementations. The attack is based on fault injection in the Cooley-Tukey butterfly operation to zeroize the twiddle factor. Therefore, the corresponding changes result in a significant reduction in the entropy of the NTT's output.

By extending this fault to an entire stage of the NTT, and eventually to the whole NTT, the output entropy is greatly reduced. This attack is also practically demonstrated against ARM Cortex-M4 implementations of Kyber and Dilithium.

Contributions In this work, we present a novel technique to protect the NTT against fault attacks based on polynomial evaluation and interpolation. Our main idea is illustrated in Fig. 1.

$$\begin{tikzcd}
f(X) \arrow[r, Rightarrow, maps to, shift left, "\mathrm{NTT}"] \arrow[d, maps to, "\text{evaluation}"'] & (f(\omega^j)) \arrow[l, Rightarrow, maps to, shift left, "\mathrm{NTT}^{-1}"] \arrow[d, maps to, "\text{interpolation}"] \\
f(u) \arrow[r, equal, "?", "\text{check}"'] & f(u)
\end{tikzcd}$$

Fig. 1. The basic idea behind our countermeasure against fault injection.

To protect the computation of NTT(f) against fault injection for some polynomial f, we suggest an implementation of the NTT that evaluates the polynomial f on a selected point u. To verify the correctness of the output of the NTT, the implementation reconstructs the value $f(u)$ by polynomial interpolation. The inverse NTT^{-1} can be protected analogously. This implies that a complete ring multiplication, i.e., NTT transformations and the point-wise multiplication, can be protected with the proposed novel technique, hence providing full protection for this operation.

We describe the details of this countermeasure, the choice of the interpolation point u, the error detection properties of the proposed countermeasure and its adaption to different forms of the NTT with application to Dilithium and Kyber. We also investigate the additional computational effort required by this countermeasure. Finally, we provide a practical evaluation of the proposed method on an ARM Cortex-M4 microcontroller. In the exemplary case of Dilithium, the results indicate that the proposed technique incurs a 72% computational overhead.

Structure This paper is structured as follows. Section 2 provides background information about the NTT, Kyber and Dilithium. Section 3 presents the proposed method for safeguarding the NTT against fault attacks, the error detection properties of the proposed countermeasure and its application to Kyber and Dilithium. Section 4 describes a practical evaluation result of the proposed countermeasures on an ARM Cortex-M4 microcontroller with application to Dilithium. Conclusions and an outlook are in Sec. 5.

2 Background

This section provides background information regarding the Number Theoretic Transform (NTT), Dilithium and Kyber. Furthermore, it fixes the notation used throughout this paper.

2.1 The Number Theoretic Transform

Let K be a field and $\phi(X) = X^n + 1$ with $n = 2^k$ for some integer $k \geq 0$. Let us assume that K contains a $2n$-th root of unity ω. Then $\phi(X)$ can be factored as follows:

$$\begin{aligned} \phi(X) &= (X^{n/2} - \omega^{n/2})(X^{n/2} - \omega^{3n/2}) \\ &= (X^{n/4} - \omega^{n/4})(X^{n/4} - \omega^{5n/4})(X^{n/4} - \omega^{3n/4})(X^{n/4} - \omega^{7n/4}) \\ &= \cdots \\ &= \textstyle\prod_{j=0}^{n-1} (X - \omega^{(2\mathrm{br}_k(j)+1)/4}), \end{aligned} \tag{1}$$

where $\mathrm{br}_k(j)$ denotes the bit-reversal of a k-bit number j, i.e., $\mathrm{br}_k\left(\sum_{i=0}^{k-1} a_i 2^i\right) = \sum_{i=0}^{k-1} a_{k-1-i} 2^i$.

The factorization of $\phi(X)$ in Eq. (1) leads to a series of ring isomorphisms over multiple layers ℓ:

$$\begin{tikzcd}
\ell = 0 : & K[X]/(X^n + 1) \arrow[d, "\cong"] \\
\ell = 1 : & K[X]/(X^{n/2} - \omega^{n/2}) \times K[X]/(X^{n/2} - \omega^{3n/2}) \arrow[d, "\cong"] \\
\vdots & \vdots \arrow[d, "\cong"] \\
\ell = k - 1 : & \prod_{j=0}^{n-1} K[X]/(X - \omega^{2\mathrm{br}_\ell(j)+1})
\end{tikzcd} \tag{2}$$

The chain of isomorphisms defined in Eq. (2) is canonical and simply given by modular reduction as follows:

$$\begin{tikzcd}
\ell = 0 : & f(X) \arrow[d, mapsto] \\
\ell = 1 : & (f(X) \bmod (X^{n/2} - \omega^{n/2}), f(X) \bmod (X^{n/2} - \omega^{3n/2})) \\
\vdots & \vdots \arrow[d, mapsto] \\
\ell = k-1 : & (f(\omega^{2\mathrm{br}_2(j)+1}))_{j=0,\dots,n-1}
\end{tikzcd} ,$$

where in the last layer $k-1$ we identify $f(X) \mod X - \omega^{2\mathrm{br}_2(j)+1}$ with $f(\omega^{2\mathrm{br}_2(j)+1})$

We define the NTT : $K[X]/(\phi) \to K^n$ as the concatenation of the isomorphisms in Eq. (2), so we have:

$$\mathrm{NTT}(f) = (f(\omega^{2\mathrm{br}_k(0)+1}), f(\omega^{2\mathrm{br}_k(1)+1}), \dots, f(\omega^{2\mathrm{br}_k(n-1)+1})). \tag{3}$$

If we equip K^n with component-wise addition and multiplication, then the NTT is a ring isomorphism. In other words, $\mathrm{NTT}(f+g) = \mathrm{NTT}(f) + \mathrm{NTT}(g)$ and $\mathrm{NTT}(f \cdot g) = \mathrm{NTT}(f) \odot \mathrm{NTT}(g)$, where $\odot$ denotes component-wise multiplication.

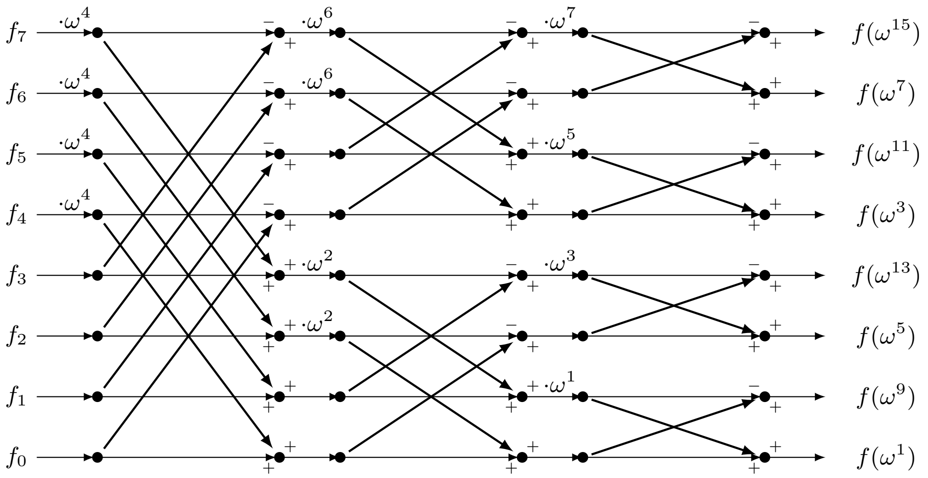

Fig. 2. The Cooley-Tukey algorithm for computing NTT.

The latter property is the reason why the NTT plays such an important role in many cryptographic schemes. It turns the computationally expensive multiplication of two polynomials of degree n into n field multiplications. This comes at the cost of first computing the NTT for the two polynomials and then computing NTT^{-1} of the component-wise product. In many applications, however, one of the polynomials is fixed, so an application can simply store and use $\mathrm{NTT}(f)$ without recomputing it every time. For this reason, many cryptographic schemes specify explicitly that the NTT of a polynomial is to be stored or transmitted to another party, rather than the polynomial in its usual representation as a string of coefficients.

2.2 Implementing the NTT

The representation of the NTT as a series of isomorphisms in Eq. (2) leads directly to an efficient implementation, namely the well-known Cooley-Tukey butterfly construction. Let $f(X)$ be a polynomial of degree $n-1$:

$$f(X) = \sum_{j=0}^{n-1} f_j X^j, \tag{4}$$

then the modular reductions mapping layer 0 to layer 1 in Eq. (2) are described by the following equations:

$$f(X) \bmod (X^{n/2} - \omega^{n/2}) = \sum_{j=0}^{n/2-1} (f_j + \omega^{n/2} f_{j+n/2}) X^j \tag{5}$$

and

$$f(X) \bmod (X^{n/2} - \omega^{3n/2}) = \sum_{j=0}^{n/2-1} (f_j - \omega^{n/2} f_{j+n/2}) X^j. \tag{6}$$

Repeating this for all layers gives the Cooley-Tukey butterfly structure of a typical NTT implementation, illustrated for $n = 8$ in Fig. 2. Reversing all operations gives the Gentleman-Sande implementation of the inverse NTT. A single butterfly does the following:

$$a' = b - a \cdot \omega^j \tag{7}$$

$$b' = b + a \cdot \omega^j \tag{8}$$

So to recover a, b from a', b' we compute:

$$a = \frac{1}{2}(b' - a')\omega^{-j} \tag{9}$$

$$b = \frac{1}{2}(b' + a') \tag{10}$$

The multiplication with $\frac{1}{2}$ can be deferred by multiplying every result by $2^{-\log_2 n}$ in a final step. Eq. (9) and (10) lead to the inverse scheme of Fig. 2 shown in Fig. 3.

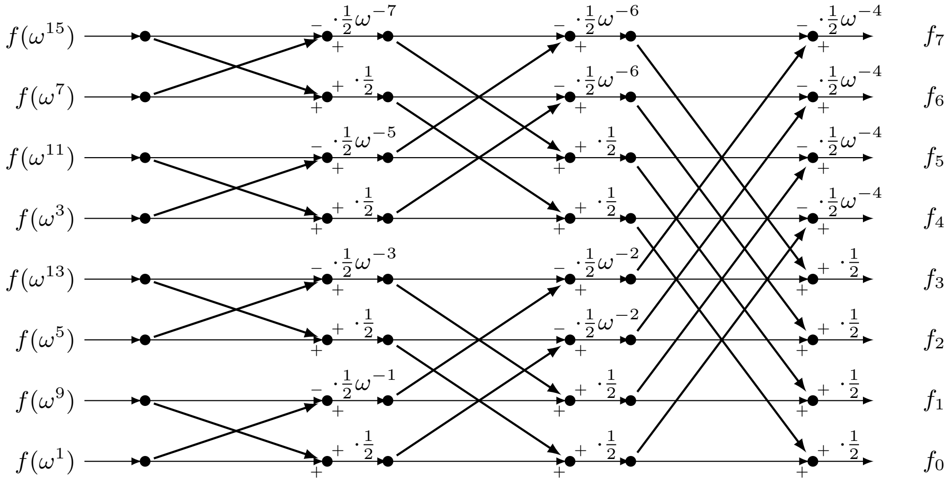

Fig. 3. The Gentleman-Sande algorithm for computing NTT^{-1}.

2.3 Dilithium

Dilithium [17] is a lattice-based general purpose digital signature scheme based on the Module Small Integer Solutions (M-SIS) and Module Learning with Errors (M-LWE) problems. The module is of dimension $k \times t$ over the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$, where $n = 256$ and $q = 2^{23} - 2^{13} + 1 = 8380417$. We see that $2n$ divides $q - 1$, so the NTT as constructed in Sec. 2 can be applied to multiply elements of $\mathcal{R}_q$. Because $n = 256 = 2^8$, it requires eight butterfly layers like the ones shown in Fig. 2. The reference implementation that is part of [17] implements the NTT in this way. For its inverse NTT^{-1} it uses the Gentleman-Sande algorithm. There are currently three versions `Dilithium-2`, `Dilithium-3`, `Dilithium-5` targeting the NIST security level 1, 3, 5, respectively. The parameters consist of the module dimension (k, t), the sampling bound of the secret η, and the rejection thresholds β and ω, cf. Table 1. The NTT is used in the key generation, signature generation, and signature verification routines of Dilithium to perform the $(k \times t) \times (t \times 1)$ matrix-to-vector polynomial multiplications $\boldsymbol{A}\boldsymbol{s}_1$, $\boldsymbol{A}\boldsymbol{y}$, and $\boldsymbol{A}\boldsymbol{z}$, respectively.

2.4 Kyber

Kyber [23] is a lattice-based key encapsulation mechanism based on the Module Learning With Errors (M-LWE) problem. The module is of dimension $t \times t$ over the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$, where $n = 256$ and $q = 13 \cdot 2^8 + 1 = 3329$. Note that n divides $q - 1$ but $2n$ does not. Therefore, the chain of isomorphisms in Eq. (2) breaks off at the penultimate seventh layer. Hence, in Kyber, the NTT reduces multiplication in $\mathcal{R}_q$ to multiplying a sequence of polynomials of degree one modulo a polynomial of degree two. The reference implementation that is part of [23] implements the NTT in this way. For its inverse NTT^{-1} it uses the Gentleman-Sande algorithm, starting at the second layer. There are currently three versions `Kyber-512`, `Kyber-768`, `Kyber-1024` targeting the NIST security levels 1, 3, 5, respectively. Each variant is specified by a parameter set, cf. Table 1, where t denotes the module dimension, (d_1, d_2) are the rounding parameters, and η is the width of the centered binomial distribution. The NTT is used in the key generation and encryption routines of Kyber to perform the $(t \times t) \times (t \times 1)$ matrix-to-vector polynomial multiplications $\boldsymbol{A}^t\boldsymbol{s}$ and $\boldsymbol{A}\boldsymbol{s}'$.

Table 1. Kyber and Dilithium parameter sets.

	NIST	t	(d_1, d_2)	$\eta(s, s')$	$\eta(e, e', e'')$
`Kyber-512`	1	2	$(10, 4)$	6	4
`Kyber-768`	3	3	$(10, 4)$	4	4
`Kyber-1024`	5	4	$(11, 5)$	4	4

	NIST	(k, t)	η	β	ω
`Dilithium-2`	1	$(4, 4)$	2	78	80
`Dilithium-3`	3	$(6, 5)$	4	196	55
`Dilithium-5`	5	$(8, 7)$	2	120	75

3 Fault Resistant NTT using Polynomial Evaluation and Interpolation Techniques

This section presents the proposed method for safeguarding the NTT against fault attacks and its error detection capability. Furthermore, it provides a calculation of the additional computational effort required and shows how to concretely use it to secure the NTT in Kyber and Dilithium.

3.1 Proposed countermeasure

The idea behind our countermeasure is shown in Fig. 1. More precisely, let $u \in K$, where the criteria for choosing u are given in Lemma 3, then our countermeasure consists of the following steps:

1. Compute $w = f(u)$ by evaluating f at u;
2. Compute $\mathrm{NTT}(f) = (f(\omega^{2\mathrm{br}_k(0)+1}), f(\omega^{2\mathrm{br}_k(1)+1}), \ldots, f(\omega^{2\mathrm{br}_k(n-1)+1}))$ with, e.g., the usual Cooley-Tukey algorithm;
3. Compute $w' = f(u)$ by interpolating the $n - 1$ output values $\mathrm{NTT}(f)$;
4. Check that $w = w'$. If this is not the case, then a fault in the computation of $\mathrm{NTT}(f)$ has been detected.

Let us first look at the computational cost of this countermeasure. Its error detection properties will be analyzed in Sec. 3.2.

Lemma 1. *Let $u \in K$ and $f \in K[X]$ of degree $n - 1$. Then computing $f(u)$ requires at most $n - 1$ multiplications and $n - 1$ additions in K.*

Proof. This refers to Horner's method. We write

$$f(X) = \sum_{j=0}^{n-1} f_j X^j = ((\cdots((f_{n-1}X + f_{n-2})X + f_{n-3})\cdots)X + f_1)X + f_0 \quad (11)$$

and count the operations on the right. □

As we have clearly shown in Eq. (3), the NTT maps a polynomial f to n values of f. By interpolation, the polynomial f can be reconstructed from these n values.

In detail, we write

$$L_j(X) = \prod_{\substack{0 \le i < n \\ i \ne j}} \frac{X - \omega^{2i+1}}{\omega^{2j+1} - \omega^{2i+1}}, \qquad j = 0, 1, \ldots, n-1 \quad (12)$$

for the n Lagrange polynomials for the points $\{\omega, \omega^3, \omega^5, \ldots, \omega^{2n-1}\}$. These polynomials form a basis of the K-vector subspace of polynomials of degree at most $n-1$ in $K[X]$ and have the property that

$$L_j(\omega^{2i+1}) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \ne j \end{cases} \quad (13)$$

Then

$$f(X) = \sum_{j=0}^{n-1} f(\omega^{2j+1}) L_j(X) \quad (14)$$

For our countermeasure, we do not want to reconstruct f from $f(\omega^{2j+1})_{j=0,\ldots,n-1}$ but just evaluate f at a single point u. We note that the values $L_j(u)$ can be precomputed as soon as u is fixed. The interpolated value can then be calculated with Eq. (14). In particular, if the point u is fixed at compile-time and only f varies at run-time, then we can precompute the values $L_j(u)$ and link them as a table to the code.

Lemma 2. *Let $u \in K$ and $f \in K[X]$ of degree $n-1$. Then computing $f(u)$ given $f(\omega^{2j+1})_{j=0,\ldots,n-1}$ and $(L_j(u))_{j=0,\ldots,n-1}$ requires at most n multiplications and $n-1$ additions.*

Proof. Count the operations on the right-hand side of Eq. (14). □

An algorithmic description of the proposed countermeasures is provided in Appendix A.

3.2 Error detection properties

Let us now have a look at the error detection capability of our countermeasure. Consider a single Cooley-Tukey butterfly as illustrated in Fig. 4:

Then let us fix an error model. We assume that fault injection can cause any of the following types of errors:

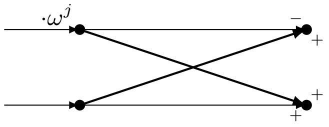

Fig. 4. A single Cooley-Tukey butterfly from Fig. 2.

1. An error in one of the input coefficients.
2. An error in the multiplication with ω^j. This is equivalent to an error in the input coefficient that is being multiplied by ω^j.
3. An error in the subtraction. This is equivalent to an error in an input coefficient in the following layer.
4. An error in the addition. This is also equivalent to an error in an input coefficient in the following layer.

We see that all four types of errors can be reduced to an error in an input coefficient in one of the layers of the Cooley-Tukey implementation of the NTT.

A fault in a single coefficient somewhere in the NTT means that the polynomial in one of the ring isomorphisms of Eq. (2) is changed. Let us assume this happens in layer ℓ and write $g(X) \in K[X]/(X^{n/2^\ell} - \omega^{(2\mathrm{br}_\ell(i)+1)n/2^\ell})$ for the affected polynomial. Then $g(X)$ is changed to

$$\tilde{g}(X) = g(X) + DX^m \tag{15}$$

for some $D \in K$ and some integer m, $0 \leq m < n/2^\ell$.

We need to determine how the error propagates through the following layers in the NTT implementation. To do this, we translate the error in layer ℓ to an error in layer 0. Define a polynomial

$$e(X) := D\Big(\prod_{j=0, j\neq i}^{2^\ell-1} \frac{X^{n/2^\ell} - \omega^{(2\mathrm{br}_\ell(j)+1)n/2^\ell}}{\omega^{(2\mathrm{br}_\ell(i)+1)n/2^\ell} - \omega^{(2\mathrm{br}_\ell(j)+1)n/2^\ell}}\Big)X^m. \tag{16}$$

Now

$$e(X) \bmod (X^{n/2^\ell} - \omega^{(2\mathrm{br}_\ell(j)+1)n/2^\ell}) = \begin{cases} DX^m & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}. \tag{17}$$

Hence, the error that replaces $g(X)$ with $\tilde{g}(X)$ is equivalent to an error that replaces the input $f(X)$ with

$$\tilde{f}(X) = f(X) + e(X) \tag{18}$$

Therefore, the injected error changes the output of the NTT to

$$\mathrm{NTT}(\tilde{f}) = \mathrm{NTT}(f) + \mathrm{NTT}(e) \tag{19}$$

and the interpolation in our countermeasure will compute $\tilde{f}(u) = f(u) + e(u)$

Lemma 3. *Let $u \in K \setminus \{0\}$ such that $u^{n/2^\ell} \neq \omega^{(2\mathrm{br}_\ell(j)+1)n/2^\ell}$ for any $0 \leq \ell \leq k$ and any $0 \leq j < 2^\ell$. Then the countermeasure described in Sec. 3.1 detects an error in a single coefficient in an NTT implementation as in Sec. 2.2.*

Proof. We have just seen that the interpolation step of the countermeasure computes $\tilde{f}(u) = f(u) + e(u)$, whereas the evaluation step computes $f(u)$. We notice from the definition of $e(X)$ in Eq. (16) that $e(u) \neq 0$. Hence, $\tilde{f}(u) \neq f(u)$ and therefore the error is detected. □

If an attacker injects several faults, we cannot provide an absolute guarantee of detecting them with our countermeasure. However, such faults are still detected with a high probability. It seems reasonable to assume that an attack with several faults will change the interpolated value randomly. In this case, the probability that an attack of this type is detected, is $1 - 1/q$ if K has q elements.

3.3 Applying the countermeasure to the inverse NTT

All the concepts presented in the previous section can be adapted to the NTT^{-1} operation and the Gentleman-Sande algorithm as well.

Specifically, the order of the operations in Sec. 3.1 changes. If the input to NTT^{-1} is $\mathrm{NTT}(f)$ for some polynomial f, then our countermeasure, applied to NTT^{-1} becomes:

1. Compute $w = f(u)$ by interpolating $\mathrm{NTT}(f)$, the input to NTT^{-1};
2. Compute $f = \mathrm{NTT}^{-1}(\mathrm{NTT}(f))$;
3. Compute $w' = f(u)$ by evaluating f on u;
4. Check that $w = w'$. If this is not the case, then a fault in the computation of NTT^{-1} has been detected.

From Eq. (9) and Eq. (10) we see that a single Gentleman-Sande butterfly looks like in Fig. 5.

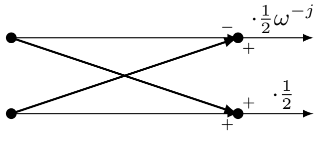

Fig. 5. A single Gentleman-Sande butterfly from Fig. 3

As in Sec. 3.2, we pointed out that the four types of errors listed there can again each be reduced to an error in a single coefficient. Such an error can again be described by the addition of a monomial DX^m as in Eq. (15). The resulting error in the output of NTT^{-1} is then provided by Eq. (16). The faulty output of NTT^{-1} is given by Eq. (18). So, in this case, the interpolation step of the countermeasure computes $f(u)$, whereas the evaluation computes $\tilde{f}(u)$. The proof of Lemma 3 shows that, if u is chosen as in Lemma 3, then $e(u) \neq 0$. Hence, $\tilde{f}(u) \neq f(u)$ and the error in the computation of NTT^{-1} is detected.

3.4 Compatibility of the countermeasure with ring operations

When used as part of a cryptographic algorithm, the purpose of the NTT is typically to accelerate multiplication. Polynomial addition, although not accelerated by the NTT, is also a common operation in cryptographic algorithms that use the NTT. Therefore, it is worth exploring the compatibility of our countermeasure with these operations and determining if it can be utilized to provide protection for them as well.

The ring multiplication $h = f \cdot g$ can efficiently be computed using the NTT as $h = \mathrm{NTT}^{-1}(\mathrm{NTT}(f \cdot g)) = \mathrm{NTT}^{-1}(\mathrm{NTT}(f) \odot \mathrm{NTT}(g))$. Therefore, our countermeasure can be extended to protect the multiplication and not just the NTT by following the steps outlined below:

1. Compute $w'_1 = f(u)$, $w'_2 = g(u)$, by interpolating the $n-1$ output values of $\mathrm{NTT}(f)$ and $\mathrm{NTT}(g)$, respectively;
2. Compute $w = h(u)$ by evaluating the result of the multiplication at u;
3. Check that $w = w'_1 w'_2$. If this is not the case, then a fault in the computation of the ring multiplication has been detected.

Note that in some algorithm specifications, e.g., Kyber, some inputs are already NTT-transformed, so that the transformations $\mathrm{NTT}(f)$ and $\mathrm{NTT}(g)$ in the first step are not always needed. For error detection to cover both the NTT operations as well as the multiplication, the checksums w'_1 and w'_2 have to be verified as described in Sec. 3.1. Otherwise, if, for example, $w'_1 = 0$, errors in the computation of $\mathrm{NTT}(g)$ may go undetected.

Analogously, our countermeasure is compatible with polynomial addition. This is particularly interesting when a polynomial is split into two shares as a countermeasure against side-channel attacks. If $f = f_1 + f_2$ and $w_1 = f_1(u)$, $w_2 = f_2(u)$, then we can check that $f(u) = w_1 + w_2$, hence providing combined side-channel and fault resistance.

3.5 Comparison with other countermeasures

An obvious way of securing an NTT implementation against single faults is to compute the NTT twice and compare the results. Computing the NTT with the Cooley-Tukey method costs $\frac{n}{2}\log_2(n)$ multiplications and $n\log_2(n)$ additions. From Lemmas 1 and 2 we see that the total cost of our countermeasure is $2n-1$ multiplications and $2n-2$ additions. Hence, the cost of our countermeasure relative to the cost of the NTT in terms of multiplications is

$$\frac{2n-1}{\frac{n}{2}\log_2(n)} = \frac{4-2/n}{\log_2(n)} \tag{20}$$

and in terms of additions it is

$$\frac{2n-2}{n\log_2(n)} = \frac{2-2/n}{\log_2(n)}. \tag{21}$$

In the case of Dilithium we have $n = 256$ and hence the cost of our countermeasure is about an extra 50% multiplications and an extra 25% additions. This is significantly less than the overhead of 100% for computing the NTT a second time. In practice, the exact performance cost depends on the implementation details (cf. Sec. 4).

In [13], a different type of countermeasure against fault attacks is presented. The authors enlarge the modulus q and use this 'extra space' to introduce redundancy into the coefficients of the NTT. The cost of this countermeasure depends very much on the hardware architecture underlying the implementation. The idea is to use registers which are wide enough to hold numbers significantly larger than q. Similarly, the effectiveness of this countermeasure depends very much on exactly this register width. An important difference between the countermeasure in [13] and the one presented in this paper is that our countermeasure guarantees the detection of a single fault in a coefficient, while the error detection property in [13] is probabilistic.

As we have seen at the end of Sec. 3.2, our countermeasure can also detect errors beyond the guaranteed detection with a certain probability. How this compares to the probabilistic error detection of [13] again depends very much on the concrete implementation. However, if we assume implementation on a 32-bit platform and if we further assume that the size of q is roughly 16 bit, then the probabilistic error detection capability of our countermeasure and that of [13] are similar.

3.6 Adapting the countermeasure to Kyber

As we have described in Sec. 2.4, the NTT in Kyber leaves out the final layer. In other words, the NTT computes $n/2$ polynomials $a_jX + b_j$ of degree one such that

$$a_jX + b_j = f(X) \bmod (X^2 - \omega^{(2\mathrm{br}_{k-1}(j)+1)2}). \tag{22}$$

So, instead of computing $f(u)$ for our countermeasure as in the previous section, it seems natural to compute $f(X) \bmod (X^2 \bmod u)$ instead.

To adapt our countermeasure, we define polynomials for $j = 0, 1, \ldots, n/2-1$:

$$M_j(X) := \prod_{i=0, i\neq j}^{2^{k-1}-1} \frac{X^2 - \omega^{(2\mathrm{br}_{k-1}(i)+1)2}}{\omega^{(2\mathrm{br}_{k-1}(j)+1)2} - \omega^{(2\mathrm{br}_{k-1}(i)+1)2}} \tag{23}$$

Lemma 4. *With the notation as above:*

$$f(X) = \sum_{j=0}^{n/2-1} (a_jX + b_j)M_j(X) \tag{24}$$

Proof. Observe that

$$M_j(X) \bmod (X^2 - \omega^{(2\mathrm{br}_{k-1}(i)+1)2}) = \begin{cases} 1 & \text{if } j = j \\ 0 & \text{if } j \neq i \end{cases}. \tag{25}$$

Hence, we have for all $i = 0, 1, \ldots, n/2 - 1$:

$$\begin{aligned} \sum_{j=0}^{n/2} (a_j X + b_j) M_j(X) \bmod (X^2 - \omega^{(2\mathrm{br}_{k-1}(i)+1)2}) &= a_i X + b_i \\ &= f(X) \bmod (X^2 - \omega^{(2\mathrm{br}_{k-1}(i)+1)2}). \end{aligned} \tag{26}$$

□

Let $u \in K$. Then, before the NTT, we can compute

$$f(X) \bmod (X^2 - u) = \Big(\sum_{j=0}^{n/2-1} f_{2j+1} u^j \Big) X + \Big(\sum_{j=0}^{n/2-1} f_{2j} u^j \Big) \tag{27}$$

Both sums can be computed efficiently with Horner's method again. This requires $n/2 - 1$ multiplications and $n/2 - 1$ additions in K for each sum. Hence, $n - 2$ multiplications and $n - 2$ additions are required in total.

Looking at the definition of $M_j(X)$ in Eq. (23), we see that $M_j(X) \bmod (X^2 - u) =: m_j \in K$ for all $j = 0, 1, \ldots, n/2 - 1$.

Hence, using Lemma 4, we can compute $f(X) \bmod (X^2 - u)$ from the NTT output, i.e., from the polynomials $a_j X + b_j$ as:

$$\begin{aligned} \sum_{j=0}^{n/2-1} (a_j X + b_j) M_j(X) \bmod (X^2 - u) &= \sum_{j=0}^{n/2-1} (a_j X + b_j) m_j \\ &= \Big(\sum_{j=0}^{n/2-1} a_j m_j \Big) X + \Big(\sum_{j=0}^{n/2-1} b_j m_j \Big) \end{aligned} \tag{28}$$

Computing the two sums on the right requires $n/2$ multiplications and $n/2 - 1$ additions for each, and so n multiplications and $n - 2$ additions in total.

Lemma 5. *Let $u \in K \setminus \{0\}$ such that $u^{n/2^{\ell+1}} \neq \omega^{(2\mathrm{br}_\ell(j)+1)n/2^\ell}$ for any $0 \leq \ell \leq k$ and any $0 \leq j < 2^\ell$. Then the countermeasure as described in this section detects an error in a single coefficient in the Kyber NTT.*

Proof. Based on the same arguments as before, any error is equivalent to an error of the type $e(X)$ as in Eq. (16). The checksum in Eq. (28) will be wrong by $e(X) \bmod (X^2 - u)$. We have chosen u such that this is non-zero. Hence, the error will be detected. □

4 Practical Evaluation

To verify the feasibility of our approach and our estimates for its performance impact, we implemented our countermeasure on a 'black pill' board with an `STM32F401CCU6` microcontroller [1]. This microcontroller is based on an ARM Cortex-M4 CPU architecture. We implemented our countermeasure for the exemplary case of Dilithium. So the field is $\mathbb{F}_q$ with $q = 8380417$, and we are working in the ring $\mathbb{F}_q[X]/(X^{256}+1)$.

We took the NTT from the Dilithium implementation by the `pqm4` library [16], a well-known library that provides optimized implementations of post-quantum cryptographic schemes for microcontroller-based platforms.

The results of our performance measurements are summarized in Table 2.

operation	clock cycles (avg.)
evaluate f	2879
interpolate NTT(f) and evaluate	3160
compute NTT(f)	8406

Table 2. Performance numbers for our countermeasure applied to Dilithium.

The relative cost of our countermeasure applied to Dilithium can easily be computed from the numbers in Table 2 as:

$$\frac{(\text{cost of evaluating } f) + (\text{cost of interpolating NTT}(f) \text{ and evaluating})}{(\text{cost of NTT})} \equiv 72\%$$

This is close to the expected overhead from the theoretical estimate given in Sec. 3.5. The implementation of our countermeasure has not been optimized for the Dilithium NTT or a particular point in the evaluation. So there may be some potential for further optimizations. The NTT implementation in the `pqm4` library, on the other hand, is highly optimized.

5 Conclusion

We have presented a countermeasure that protects an implementation of the NTT or its inverse against a single fault in one of the coefficients. We have seen that this fault model also covers faults in a twiddle factor, the multiplication with a twiddle factor and the addition in a butterfly operation. Our countermeasure requires $2n-1$ multiplications and $2n-2$ additions in the field K, hence is significantly faster than a redundant computation. We have also shown how to adapt our countermeasure to situations where the computation of the NTT is 'incomplete', as it is the case for Kyber. Our countermeasure can be safely combined with further masking and shuffling countermeasures to achieve combined fault and side-channel protections. Finally, it is worth noting that the countermeasure presented in this paper can also be applied to other schemes using the NTT operation, e.g., other cryptographic schemes based on structured lattices.

Acknowledgments. This work was partly funded by the German Federal Ministry for Economic Affairs and Climate Action in the project PoQsiKom through grant number 13I40V010 and by the German Federal Ministry of Education and Research in the project Quoryptan through grant number 16KIS2033.

References

1. WeAct Black Pill V3.0, https://stm32-base.org/boards/STM32F401CEU6-WeAct-Black-Pill-V3.0.html, accessed 2023-12-21
2. Fips 203 (draft) module-lattice-based key-encapsulation mechanism standard. Tech. rep. (Aug 2023). https://doi.org/10.6028/NIST.FIPS.203.ipd, https://doi.org/10.6028/NIST.FIPS.203.ipd
3. Fips 204 (draft) module-lattice-based digital signature standard. Tech. rep. (Aug 2023). https://doi.org/10.6028/NIST.FIPS.204.ipd, https://doi.org/10.6028/NIST.FIPS.204.ipd
4. Bauer, S., De Santis, F.: A differential fault attack against deterministic falcon signatures. IACR Cryptol. ePrint Arch. p. 422 (2023), https://eprint.iacr.org/2023/422
5. Bauer, S., De Santis, F.: Forging dilithium and falcon signatures by single fault injection. In: 2023 Workshop on Fault Detection and Tolerance in Cryptography (FDTC). IEEE Computer Society (2023)
6. Berthet, P., Tavernier, C., Danger, J., Sauvage, L.: Quasi-linear masking to protect kyber against both SCA and FIA. IACR Cryptol. ePrint Arch. p. 1220 (2023), https://eprint.iacr.org/2023/1220
7. Bindel, N., Buchmann, J., Krämer, J.: Lattice-based signature schemes and their sensitivity to fault attacks. Cryptology ePrint Archive, Report 2016/415 (2016), https://eprint.iacr.org/2016/415
8. Bos, J.W., Gourjon, M., Renes, J., Schneider, T., van Vredendaal, C.: Masking kyber: First- and higher-order implementations. IACR TCHES **2021**(4), 173–214 (2021). https://doi.org/10.46586/tches.v2021.i4.173-214, https://tches.iacr.org/index.php/TCHES/article/view/9064
9. Bruinderink, L.G., Pessl, P.: Differential fault attacks on deterministic lattice signatures. IACR TCHES **2018**(3), 21–43 (2018). https://doi.org/10.13154/tches.v2018.i3.21-43, https://tches.iacr.org/index.php/TCHES/article/view/7267
10. Coron, J., Gérard, F., Trannoy, M., Zeitoun, R.: Improved gadgets for the high-order masking of dilithium. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(4), 110–145 (2023). https://doi.org/10.46586/TCHES.V2023.I4.110-145, https://doi.org/10.46586/tches.v2023.i4.110-145
11. Delvaux, J.: Roulette: A diverse family of feasible fault attacks on masked kyber. IACR TCHES **2022**(4), 637–660 (2022). https://doi.org/10.46586/tches.v2022.i4.637-660
12. Delvaux, J., Merino Del Pozo, S.: Roulette: Breaking kyber with diverse fault injection setups. Cryptology ePrint Archive, Report 2021/1622 (2021), https://eprint.iacr.org/2021/1622

13. Heinz, D., Pöppelmann, T.: Combined fault and DPA protection for lattice-based cryptography. Cryptology ePrint Archive, Report 2021/101 (2021), https://eprint.iacr.org/2021/101
14. Hermelink, J., Pessl, P., Pöppelmann, T.: Fault-enabled chosen-ciphertext attacks on kyber. In: Adhikari, A., Küsters, R., Preneel, B. (eds.) Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13143, pp. 311–334. Springer (2021). https://doi.org/10.1007/978-3-030-92518-5_15, https://doi.org/10.1007/978-3-030-92518-5_15
15. Hermelink, J., Streit, S., Strieder, E., Thieme, K.: Adapting belief propagation to counter shuffling of ntts. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(1), 60–88 (2023). https://doi.org/10.46586/TCHES.V2023.I1.60-88, https://doi.org/10.46586/tches.v2023.i1.60-88
16. Kannwischer, M.J., Petri, R., Rijneveld, J., Schwabe, P., Stoffelen, K.: PQM4: Post-quantum crypto library for the ARM Cortex-M4, https://github.com/mupq/pqm4
17. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S.: CRYSTALS-DILITHIUM. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
18. Migliore, V., Gérard, B., Tibouchi, M., Fouque, P.: Masking dilithium - efficient implementation and side-channel evaluation. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11464, pp. 344–362. Springer (2019). https://doi.org/10.1007/978-3-030-21568-2_17, https://doi.org/10.1007/978-3-030-21568-2_17
19. NIST: NIST announces first four quantum-resistant cryptographic algorithms. https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms (2022), accessed 2022-12-21
20. Ravi, P., Chattopadhyay, A., Baksi, A.: Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Survey and new results. Cryptology ePrint Archive, Report 2022/737 (2022), https://eprint.iacr.org/2022/737
21. Ravi, P., Poussier, R., Bhasin, S., Chattopadhyay, A.: 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: Batina, L., Picek, S., Mondal, M. (eds.) Security, Privacy, and Applied Cryptography Engineering - 10th International Conference, SPACE 2020, Kolkata, India, December 17-21, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12586, pp. 123–146. Springer (2020). https://doi.org/10.1007/978-3-030-66626-2_7, https://doi.org/10.1007/978-3-030-66626-2_7
22. Ravi, P., Yang, B., Bhasin, S., Zhang, F., Chattopadhyay, A.: Fiddling the twiddle constants - fault injection analysis of the number theoretic transform. IACR TCHES **2023**(2), 447–481 (2023). https://doi.org/10.46586/tches.v2023.i2.447-481
23. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D., Ding, J.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022

24. Xagawa, K., Ito, A., Ueno, R., Takahashi, J., Homma, N.: Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 33–61. Springer, Heidelberg (Dec 2021). https://doi.org/10.1007/978-3-030-92075-3_2

A Algorithmic Countermeasure

This section provides an algorithmic description of the proposed fault countermeasure to protect the NTT operation. Alg. 1 describes a fault resistant NTT for $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$ and $K = \mathbb{Z}_q$, while Alg. 2 and Alg. 3 describe algorithms for polynomial evaluation and interpolation using the Horner and Lagrange techniques, respectively. In particular, Alg. 3 takes advantage of a precomputation algorithm specified in Alg. 4.

Algorithm 1 Algorithmic description of the fault resistant NTT for $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$ and $K = \mathbb{Z}_q$.

```
Require: $f \in \mathcal{R}_q$ with $f = (f_0, ..., f_{n-1})$, $u \in K$ as defined in Lem. 3 and $L = \textsc{Precompute}(u)$
Ensure: $\hat{f} \in K^n$ s.t. $\hat{f} = (\hat{f}_0, ..., \hat{f}_{n-1})$ with $\hat{f}_j = f(\omega^{2\mathrm{br}_k(j)+1})$ for $0 \le j < n-1$
procedure FaultResistant-NTT($f, u, L$)
    $w \leftarrow \textsc{Eval}(f, u)$
    $\hat{f} \leftarrow \mathrm{NTT}(f)$
    $w' \leftarrow \textsc{Interpolate}(\hat{f}, L)$
    if $w \neq w'$ then
        Error()
    end if
    return $\hat{f} = (\hat{f}_0, ..., \hat{f}_{n-1})$
end procedure
```

Algorithm 2 Evaluation by Horner's rule for $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$ and $K = \mathbb{Z}_q$.

```
Require: $f \in \mathcal{R}_q$ s.t. $f = (f_0, ..., f_{n-1})$ and $u \in K$
Ensure: $w \in K$
procedure Eval($f, u$)
    $w \leftarrow f_{n-1}$
    for $i \leftarrow 0$ to $n-2$ do
        $w \leftarrow f_{n-2-i} + wu$
    end for
    return $w$
end procedure
```

Algorithm 3 Lagrange interpolation with immediate evaluation for $K = \mathbb{Z}_q$.

```
Require: $\hat{f} \in K^n$ with $\hat{f} = (\hat{f}_0, ..., \hat{f}_{n-1})$, $u \in K$ and $L = \textsc{Precompute}(u)$
Ensure: $w' \in K$
procedure $\textsc{Interpolate}(\hat{f}, L)$
    $w' \leftarrow 0$
    for $i \leftarrow 0$ to $n - 1$ do
        $w' \leftarrow w' + \hat{f}_i \cdot L[i]$
    end for
    return $w'$
end procedure
```

Algorithm 4 Precompute the $L_i(u)$ for interpolation, where L_i is defined in Eq. (12) and $K = \mathbb{Z}_q$.

```
Require: $u \in K$
Ensure: $L = (L_0(u), L_1(u), \ldots, L_{n-1}(u))$
procedure $\textsc{Precompute}(u)$
    for $i \leftarrow 0$ to $n - 1$ do
        $L[i] \leftarrow L_i(u)$
    end for
    return $L$
end procedure
```