RESEARCH-ARTICLE

Efficient Algorithm-Level Error Detection for Number-Theoretic Transform Used for Kyber Assessed on FPGAs and ARM

KASRA AHMADI, University of South Florida, Tampa, Tampa, FL, United States

SAEED AGHAPOUR, University of South Florida, Tampa, Tampa, FL, United States

MEHRAN MOZAFFARI KERMANI, University of South Florida, Tampa, Tampa, FL, United States

REZA AZARDERAKHSH, Florida Atlantic University, Boca Raton, FL, United States

Open Access Support provided by:

University of South Florida, Tampa

Florida Atlantic University

Published: 13 September 2025
Online AM: 15 August 2025
Accepted: 11 August 2025
Revised: 28 April 2025
Received: 18 September 2024

Efficient Algorithm-Level Error Detection for Number-Theoretic Transform Used for Kyber Assessed on FPGAs and ARM

KASRA AHMADI, Bellini College of AI, Cybersecurity, and Computing, University of South Florida, Tampa, United States
SAEED AGHAPOUR, Bellini College of AI, Cybersecurity, and Computing, USF, Tampa, United States
MEHRAN MOZAFFARI KERMANI, Bellini College of AI, Cybersecurity, and Computing, University of South Florida, Tampa, United States
REZA AZARDERAKHSH, Florida Atlantic University, Boca Raton, United States

Polynomial multiplication stands out as a highly demanding arithmetic process in the development of post-quantum cryptosystems. The importance of the number-theoretic transform (NTT) extends beyond post-quantum cryptosystems, proving valuable in enhancing existing security protocols such as digital signature schemes and hash functions. CRYSTALS-KYBER stands out as the sole public key encryption (PKE) algorithm chosen by the National Institute of Standards and Technology (NIST) in its third round selection, making it highly regarded as a leading post-quantum cryptography (PQC) solution. Faults have the potential to disrupt cryptographic systems, compromise data integrity, and enable side-channel attacks, making the incorporation of robust error detection mechanisms essential. This article introduces algorithm-level fault detection schemes in the NTT multiplication using Negative Wrapped Convolution (NWC) and the NTT tailored for Kyber Round 3, representing a significant enhancement compared with previous research. We evaluate this through the simulation of a fault model, ensuring that the conducted assessments accurately mirror the obtained results. Our fault detection scheme is designed to address both malicious fault injection attacks on Kyber and naturally occurring faults. Furthermore, we assessed the effectiveness of the proposed error detection scheme for the NTT implemented in both NWC and Kyber, using AMD/Xilinx Artix-7 FPGA, HLS and processor-based approaches. In our FPGA implementation of NWC, the integration of our error detection approach achieves near-100% fault coverage with minimal area overhead and results in only a 12% increase in latency compared with the original hardware design. Finally, we attained an error detection ratio of nearly 100% for the NTT operation in Kyber, with a clock cycle overhead of 16% on the Cortex-A72 processor.

CCS Concepts: • **Security and privacy → Hardware attacks and countermeasures**; *Information-theoretic techniques;*

Additional Key Words and Phrases: Fault detection, fast fourier transform (FFT), kyber, negative wrapped convolution, number-theoretic transform (NTT)

This work was supported by the US National Science Foundation (NSF) through the award SaTC-2350213.
Authors' Contact Information: Kasra Ahmadi, Bellini College of AI, Cybersecurity, and Computing, University of South Florida, Tampa, Florida, United States; e-mail: ahmadi1@usf.edu; Saeed Aghapour, Bellini College of AI, Cybersecurity, and Computing, USF, Tampa, Florida, United States; e-mail: aghapour@usf.edu; Mehran Mozaffari Kermani, Bellini College of AI, Cybersecurity, and Computing, University of South Florida, Tampa, Florida, United States; e-mail: mehran2@usf.edu; Reza Azarderakhsh, Florida Atlantic University, Boca Raton, Florida, United States; e-mail: razarderakhsh@fau.edu.

ACM 1539-9087/2025/09-ART79
https://doi.org/10.1145/3762186

ACM Reference Format:
Kasra Ahmadi, Saeed Aghapour, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2025. Efficient Algorithm-Level Error Detection for Number-Theoretic Transform Used for Kyber Assessed on FPGAs and ARM. *ACM Trans. Embedd. Comput. Syst.* 24, 5, Article 79 (September 2025), 23 pages. https://doi.org/10.1145/3762186

1 Introduction

Fast Fourier Transform (**FFT**) [12] algorithms which are used to compute the **Discrete Fourier Transform** (**DFT**) have various applications, ranging from digital signal processing to the efficient multiplication of large integers. When the coefficients of the polynomial are specifically chosen from a finite field, the resulting transform is known as the **Number Theoretic Transform** (**NTT**) [13], and it can be computed using FFT algorithms designed for operations within this particular finite field. An efficient approach for polynomial multiplication, NTT holds significant importance in post-quantum cryptosystems, e.g., the lattice-based cryptosystems which are regarded as a leading contender for quantum-secure public key cryptography, primarily because of its broad applicability and security proofs grounded in the worst-case hardness of established lattice problems.

In 2022, NIST unveiled the Round 3 outcomes of the Post-Quantum Cryptography standardization Process, which featured four chosen algorithms (CRYSTALS-KYBER , CRYSTALS-DILITHIUM, Falcon, SPHINCS+). These standards are formally issued after August 2024 with respect to **Federal Information Processing Standards** (**FIPS**) as FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard, FIPS 204, Module-Lattice-Based Digital Signature Standard, and FIPS 205, Stateless Hash-Based Digital Signature Standard [40–42]. Falcon is also formally numbered as FIPS 206. Out of the chosen algorithms, CRYSTALS-KYBER, also referred to as Kyber [8, 40], stands out as the sole algorithm for both public key encryption and key establishment that relies on the challenge of the **module learning with errors** (**MLWE**) problem.

The NTT has proven to be a potent tool that facilitates the computation of this operation with quasi-linear complexity $O(n.lgn)$. Several recent studies have been conducted on optimizing the NTT [7, 18, 23, 24, 31, 32, 38, 54, 56]. In addition to polynomial multiplication, the use of the NTT has the potential to significantly enhance existing schemes by improving their security parameters. NTT is widely used in signature schemes, hash functions, and identification schemes. As a result, incorporating efficient error detection mechanisms in the NTT for polynomial multiplication will not only improve the security and reliability but also mitigate the risk of fault attacks in the respective algorithms in post-quantum cryptography. Moreover, such schemes could at least alleviate the attack surface or be add-ons to other approaches. Kyber and Dilithium employ polynomial multiplication over $\mathbb{Z}[X]/(X^n + 1)$ utilizing the NTT. Integrating effective error detection schemes into the NTT used in Kyber will enhance security, reliability, and mitigate the risk of fault attacks in post-quantum cryptography algorithms.

1.1 Related Works

Several previous studies have concentrated on the implementation and fault detection in different arithmetic components of both classical and post-quantum cryptography [10, 19, 33, 47, 51]. We categorize relevant studies into three subsets. The initial category centers on fault attacks and countermeasures applied to post-quantum cryptographic schemes, particularly those extensively utilizing the NTT. Numerous prior studies have explored fault attacks with the objective of gaining access to or compromising secret keys [16, 21, 22, 44, 49, 55]. Ravi et al. [46] presented the first fault injection analysis of the NTT. The authors found a significant flaw in the way the NTT is implemented in the pqm4 library [27]. The identified vulnerability, referred to as twiddle-pointer, is exploited to demonstrate practical and effective attacks on Kyber and Dilithium [4]. In [45],

the authors proposed a fault attack targeting the NTT operation during the key generation and encapsulation routine of the Kyber. The fault attack is achieved by zeroizing all twiddle factors used in the NTT operation. When focusing on applying this technique to the NTT regarding secrets or errors, it can significantly diminish the randomness of the secret or error and reduce the entropy. Several other authors have reported employing fault injection on structured lattice-based schemes as a foundation for attacks [6, 9, 39] threatening the security of implementations. Regarding active attacks, although some prior research addressed fault injection, Espitau et al. [17] introduced loop-abort faults in several lattice-based cryptosystems, including CRYSTALS-Kyber. In this attack, a fault is injected into the cryptosystem, causing the loop responsible for sampling random Gaussian secret coefficients to terminate early. This early termination leads to the generation of lower-dimensional secrets, which can then be leveraged for a key recovery attack. In 2021, Pessl and Prokop [43] proposed an attack that involves a single instruction-skipping fault during the decoding process. Their fault simulations showed that at least 6,500 faulty decapsulation attempts are needed to fully recover the key for Kyber512 operating on a Cortex M4. In the same year, Hermelink et al. [22] combined fault injections with chosen-ciphertext attacks on CRYSTALS-Kyber, suggesting that their attack could bypass defenses such as decoder shuffling. Their findings demonstrated successful secret key recovery using 7,500 inequalities for Kyber-512, 10,500 for Kyber-768, and 11,000 for Kyber-1024. Delvaux [14] refined the **side-channel attack** (**SCA**) from [22], making it simpler to execute and more challenging to defend against. In [30], the authors proposed a new fault attack on the SCA-secure masked decapsulation algorithm for generic LWE-based KEMs and detailed the attack for Kyber. Jendral [25] presented an attack on CRYSTALS-Dilithium implementation on ARM Cortex-M4 using fault injection. Krahmer et al. [29] presented two key-recovery fault attacks on Dilithium's signing procedure.

The focus of the second category lies in offering fault detection methodologies pertaining to the algorithmic level of classical cryptographic schemes. In Reference [2], the authors suggested an effective algorithmic-level error detection for the ECSM window method, aiming to identify both permanent and transient errors. In Reference [1], the authors presented algorithm level error detection scheme designed for Montgomery ladder ECSM algorithm utilized in non-supersingular elliptic curves.

The third category focuses on developing fault detection schemes specifically for the NTT. In [5], the authors introduced a method for safeguarding the NTT against fault attacks. In Reference [53], the authors integrate the advantages of bit slicing, a software implementation technique where a data-path of an n-bit processor is treated as n parallel single-bit data-paths, to devise a fault countermeasure for the NTT used in Dilithium [15]. Sarker et al. [50] and [52], presented error detection architectures of the NTT based on recomputation with encoded operands. The authors achieved high error coverage with lower area and latency overhead. However, due to the recomputation process, their latency is doubled. We note that despite high error coverage, for fast implementations, these works might increase the total time to levels not acceptable. Additionally, Cintas-Canto et al. [11] introduced error detection schemes for lattice-based KEMs using recomputation techniques and implemented these schemes on FPGA. Below, we present our major contributions in this work.

1.2 Our Major Contributions

— We have proposed an algorithm-level fault detection scheme of the NTT multiplication using `NWC` which is widely used in lattice-based cryptography. We achieved high error coverage with less area overhead and latency compared with previous works.
— As the `NWC` method does not work for the NTT multiplication used in `Kyber` Round 3, we have introduced an algorithm-level error detection scheme for the NTT multiplication tailored for `Kyber` Round 3.

— We performed simulations for single and burst fault injection using our proposed schemes. The simulation outcomes demonstrated that our approach is capable of detecting diverse types of faults with high error coverage and offers protection against the NTT fault attack presented in [46] and those with respective fault models.
— We evaluate the efficiency of our proposed error detection scheme applied to the NTT within both NWC and Kyber, targeting implementations on AMD/Xilinx Artix-7 FPGA, HLS, and processor-based platforms. For the NWC hardware implementation on FPGA, our integrated error detection mechanism provides nearly 100% fault coverage, introducing only a minor increase in area and a 12% rise in latency compared with the original design. Additionally, for the Kyber NTT execution on the Cortex-A72 processor, our method achieves close to full error detection with a clock cycle overhead of 16%.

Our work is structured as follows: Section 2 discusses the NTT module used in NWC and Kyber. The presented error detection schemes are described in Section 3. We introduced the fault attack against Kyber and compare our method to other methods in Section 4. Section 5 is divided into two subsections. First, the fault model in this work is discussed; next, the fault simulation is performed to assess error detection rate. In Section 6, we integrate the proposed fault detection schemes into the hardware implementations of NTT used in Kyber and NWC to evaluate the associated overheads in area and latency. HLS and processor-based implementations are described in Section 7. Finally, our conclusions are presented in Section 8.

2 Preliminaries

2.1 Number-Theoretic Transform (NTT)

Multiplication of two polynomials f and g takes quadratic complexity $O(n^2)$ utilizing school book algorithm. However, by using the NTT, it can be reduced to quasi-linear complexity $O(n.lgn)$. The NTT represents a specialized version of the DFT, utilizing a coefficient ring chosen from a finite field that encompasses the necessary roots of unity. We denote the ring $\mathbb{Z}[X]/(X^n+1)$ by R and the ring $\mathbb{Z}_q[X]/(X^n+1)$ by R_q. Consider ω to be a primitive n-th root of unity in $\mathbb{Z}_q$ which $\omega^n \equiv 1 \pmod q$ where $q \equiv 1 \pmod{2n}$, q is a prime number, and n is a power of 2. The NTT of a polynomial $f \in R_q$ is defined as $\hat{f} = \mathrm{NTT}(f) = \sum_{j=0}^{n-1} f[j]\omega^{ij} \pmod q$ and the inverse NTT $f = \mathrm{NTT}^{-1}(\hat{f}) = \sum_{j=0}^{n-1} n^{-1}\hat{f}[j]\omega^{-ij} \pmod q$. The NTT of a sequence f is derived as in the form of matrix multiplication described in (1). The Cooley-Tukey butterfly algorithm can be used to efficiently implement the forward NTT.

$$\hat{f} = \mathrm{NTT}(f) = \begin{bmatrix} \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^1 & \dots & \omega^{n-1} \\ \omega^0 & \omega^2 & \dots & \omega^{2(n-1)} \\ . & . & \dots & . \\ \omega^0 & \omega^{n-1} & \dots & \omega^{(n-1)^2} \end{bmatrix} * \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ . \\ f(n-1) \end{bmatrix}. \tag{1}$$

In Algorithm 1, the iterative NTT implementation derives the NTT of a specified polynomial f. The bit_reverse(k) function (presented in Line 5) rearranges the input k where the new placement of elements is determined by reversing the binary representation of the operand. Lines 9 and 10 perform the butterfly operation. Every butterfly module, taking inputs a and b and producing outputs c and d, executes the following butterfly computation: $c = a + b\omega^k$ and $d = a - b\omega^k$. Kyber's reference implementation uses Montgomery multiplication to achieve efficient modular multiplication and improve performance by converting numbers into Montgomery form.

ALGORITHM 1: Iterative NTT [57]

Input: $f \in \mathbb{Z}_q$ of length $n = 2^k$, ω is primitive n-th root of unity

Output: $\hat{f} = \text{NTT}(f)$ in bit-reversed order

$\hat{f} = f$
for $s = k$ **to** 1:
    $m = 2^s$
    for $k = 0$ **to** $2^{k-s} - 1$:
        $\omega = \omega^{\textbf{bit-reverse}(k)\,.\,m/2}$
        for $j = 0$ **to** $m/2 - 1$:
            $u = \hat{f}[k.m + j]$
            $t \equiv \omega\,.\,\hat{f}[k.m + j + m/2] \bmod q$
            $\hat{f}[k.m + j] \equiv (u + t) \bmod q$
            $\hat{f}[k.m + j + m/2] \equiv (u - t) \bmod q$
return $\hat{f}$

ALGORITHM 2: The Pre-Process Step for the NTT Calculations

Input: $f \in \mathbb{Z}_q$ of length n, ψ primitive $2n$-th root of unity

Output: $\tilde{f} = \text{pre-process}(f)$

for $i = 0$ **to** $n - 1$:
    $\tilde{f}[i] = f[i]\,.\,\psi^i$
return $\tilde{f}$

ALGORITHM 3: The Post-Process Step for the NTT Calculations

Input: $f \in \mathbb{Z}_q$ of length n, ψ primitive $2n$-th root of unity

Output: $\tilde{f} = \text{post-process}(f)$

for $i = 0$ **to** $n - 1$:
    $\tilde{f}[i] = f[i]\,.\,\psi^{-i}$
return $\tilde{f}$

2.2 Negative Wrapped Convolution (NWC)

Let $f, g \in R_q$. Computing $h = f.g \bmod X^n + 1$ requires applying the NTT of length $2n$ and n zeros to be extended to f and g. This essentially doubles the input length and also necessitates an explicit reduction modulo $X^n + 1$. This problem can be resolved by utilizing NWC, which prevents the doubling of input length [34]. Let ψ be a primitive $2n$th root of unity in $\mathbb{Z}_q$ where $\psi^2 = \omega$.

Moreover, there exists a Pre-process module which is described in Algorithm 2. NWC of f and g is defined as $h = \text{post-process}(\text{INTT}(\text{NTT}(\tilde{f}) \circ \text{NTT}(\tilde{g})))$ where $\tilde{f} = \text{pre-process}(f)$, $\tilde{g} = \text{pre-process}(g)$, and the symbol $\circ$ represents component-wise multiplication. We can achieve Post-process function by changing ψ to ψ^{-1}. Post-process function is described in Algorithm 3.

2.3 NTT in Kyber

To use NWC, we require that $2n \mid (q - 1)$. Regarding the parameters used in Kyber with the prime $q = 3329$ and $n = 256$ the base field $\mathbb{Z}_q$ contains 256-th roots of unity but not 512-th roots. Therefore, we cannot use the method used in [34]. To overcome this problem, the polynomial $X^{256} + 1$ of

ALGORITHM 4: NTT in Kyber

Input: $r \in \mathbb{Z}_q$ of length $n = 256$, $\omega = 17$ is primitive n-th root of unity
Output: NTT(r) in bit-reversed order

1: $\tilde{r} = r$
2: $j = 0$
3: $k = 0$
4: **for** ($s = 128; s >= 2; s = s/2$):
5: **for** ($i = 0; i < 256; i = j + s$):
6: $zeta = \omega^{(\textbf{bit-reverse}(k))}$
7: $k = k + 1$
8: **for** ($j = i; j < i + s; j++$)
9: $t = \text{fqmul}(zeta, \tilde{r}[j + s])$
10: $\tilde{r}[j + s] = \tilde{r}[j] - t \bmod q$
11: $\tilde{r}[j] = \tilde{r}[j] + t \bmod q$
12: **return** $\tilde{r}$

R factor into 128 polynomials of degree 2 modulo q. The polynomial $X^{256} + 1$ can be written as

$$X^{256} + 1 = \prod_{i=0}^{127}(X^2 - \omega^{2i+1}) = \prod_{i=0}^{127}(X^2 - \omega^{2b_7(i)+1}),$$

where $b_7(i)$ is the bit reversal of the 7-bit i. Therefore, the NTT of f is a vector of 128 polynomials of degree one.

$$\hat{f} = \text{NTT}(f) = (\hat{f}_0 + \hat{f}_1 X, \hat{f}_2 + \hat{f}_3 X, ..., \hat{f}_{254} + \hat{f}_{255} X),$$

with

$$\hat{f}_{2i} = \sum_{j=0}^{127} f_{2j}\omega^{(2br_7(i)+1)j}, \tag{2}$$

$$\hat{f}_{2i+1} = \sum_{j=0}^{127} f_{2j+1}\omega^{(2br_7(i)+1)j}. \tag{3}$$

We can compute $h = \hat{f} \circ \hat{g} = \text{NTT}^{-1}(\text{NTT}(f) \circ \text{NTT}(g))$ consisting of the 128 products of linear polynomials

$$\hat{h}_{2i} + \hat{h}_{2i+1} X = (\hat{f}_{2i} + \hat{f}_{2i+1} X)(\hat{g}_{2i} + \hat{g}_{2i+1} X) \bmod (X^2\text{-}\omega^{2b_7(i)+1}). \tag{4}$$

The iterative NTT implementation of Kyber is described in Algorithm 4. the fqmul function (presented in Line 9 of Algorithm 4) performs multiplication of two operands and Montgomery reduction afterward.

3 Proposed Error Detection Scheme

3.1 Negative Wrapped Convolution

In this section, we present our error detection schemes on the sub-block in the NWC that uses the NTT and pre-process modules, $\text{NTT}(\tilde{f}) \bigcirc \text{NTT}(\tilde{g})$ where $\tilde{f} = \text{pre-process}(f)$ and $\tilde{g} = \text{pre-process}(g)$. We have devised an error detection scheme at the algorithm level on the component-wise NTT multiplication sub-block, $\text{NTT}(\tilde{f}) \bigcirc \text{NTT}(\tilde{g})$, as well as an error detection scheme for the pre-process function, involving utilization of recomputation using shifted operands. This error detection scheme can be applied to designs which require polynomial multiplication using the NTT.

3.1.1 Component-Wise NTT Multiplication. Our focus is on $\mathrm{NTT}(\tilde{f}) \bigcirc \mathrm{NTT}(\tilde{g})$ operation in this section. Jou et al. [26] proposed an algorithm level error detection scheme on FFT network. Their proposed error detection scheme is depicted in Figure 1. In order to achieve an efficient scheme, we have adapted the aforementioned approach to the component-wise NTT multiplication. The primary focus is to use an error detection scheme with small overhead and high error detection ratio with respect to efficient realizations of the NTT. From the NTT definition, we can use the relations presented below:

$$\hat{f}(0) = \sum_{j=0}^{n-1} \tilde{f}(j)\omega^0 = \sum_{j=0}^{n-1} \tilde{f}(j), \quad j = 0, 1, \ldots, n-1, \tag{5}$$

$$\hat{g}(0) = \sum_{j=0}^{n-1} \tilde{g}(j)\omega^0 = \sum_{j=0}^{n-1} \tilde{g}(j), \quad j = 0, 1, \ldots, n-1. \tag{6}$$

The result of the NTT multiplication for index 0 can be written as

$$\hat{f}(0)\hat{g}(0) = \sum_{j=0}^{n-1} \tilde{f}(j) \sum_{j=0}^{n-1} \tilde{g}(j). \tag{7}$$

We note that Equation (7) is used in our proposed error detection scheme. If we rotate the input sequence of Equation (1) by one, then we get:

$$\begin{bmatrix} \hat{f}(0) \\ \hat{f}(1) \\ \hat{f}(2) \\ . \\ \hat{f}(N-1) \end{bmatrix} = \rho * \begin{bmatrix} \tilde{f}(1) \\ \tilde{f}(2) \\ . \\ \tilde{f}(N-1) \\ \tilde{f}(0) \end{bmatrix} \text{or}$$

$$\begin{bmatrix} \hat{f}(0)/\omega^0 \\ \hat{f}(1)/\omega^1 \\ \hat{f}(2)/\omega^2 \\ . \\ \hat{f}(n-1)/\omega^{(n-1)} \end{bmatrix} = \theta * \begin{bmatrix} \tilde{f}(1) \\ \tilde{f}(2) \\ . \\ \tilde{f}(n-1) \\ \tilde{f}(0) \end{bmatrix} \text{where the symbol } * \text{ denotes matrix multiplication,}$$

$$\rho = \begin{bmatrix} \omega^0 & \omega^0 & \ldots & \omega^0 \\ \omega^1 & \omega^2 & \ldots & \omega^0 \\ \omega^2 & \omega^4 & \ldots & \omega^0 \\ . & . & \ldots & . \\ \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^0 \end{bmatrix} \text{and} \theta = \begin{bmatrix} \omega^0 & \omega^0 & \ldots & \omega^0 \\ \omega^0 & \omega^1 & \ldots & \omega^{n-1} \\ \omega^0 & \omega^2 & \ldots & \omega^{2(n-1)} \\ . & . & \ldots & . \\ \omega^0 & \omega^{n-1} & \ldots & \omega^{(n-1)^2} \end{bmatrix}.$$

$$\text{From } \theta * \begin{bmatrix} \alpha\tilde{f}(0) \\ \alpha\tilde{f}(1) \\ \alpha\tilde{f}(2) \\ . \\ \tilde{\alpha f}(n-1) \end{bmatrix} + \theta * \begin{bmatrix} \tilde{\beta f}(1) \\ \beta\tilde{f}(2) \\ . \\ \tilde{\beta f}(n-1) \\ \tilde{\beta f}(0) \end{bmatrix}, \text{ we get the following:}$$

$$\begin{bmatrix} \hat{f}(0)(\alpha + \beta\omega^{-0}) \\ \hat{f}(1)(\alpha + \beta\omega^{-1}) \\ \hat{f}(2)(\alpha + \beta\omega^{-2}) \\ . \\ \hat{f}(n-1)(\alpha + \beta\omega^{-(n-1)}) \end{bmatrix} = \theta * \begin{bmatrix} \alpha\tilde{f}(0) + \beta\tilde{f}(1) \\ \alpha\tilde{f}(1) + \beta\tilde{f}(2) \\ . \\ \alpha\tilde{f}(n-2) + \beta\tilde{f}(n-1) \\ \alpha\tilde{f}(n-1) + \beta\tilde{f}(0) \end{bmatrix}, \tag{8}$$

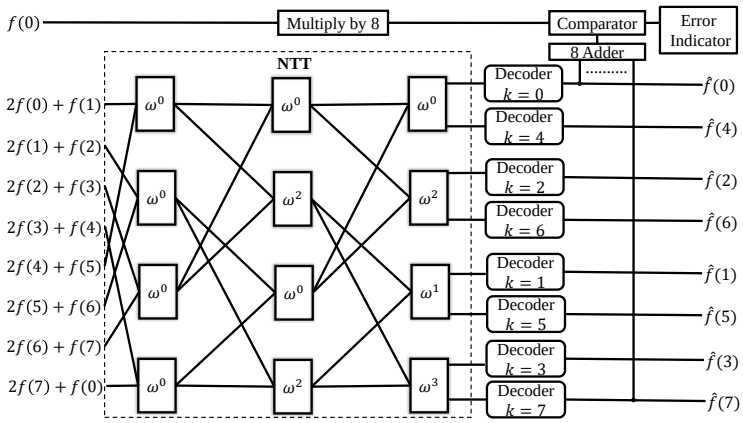

Fig. 1. Concurrent error detection scheme for the NTT operation.

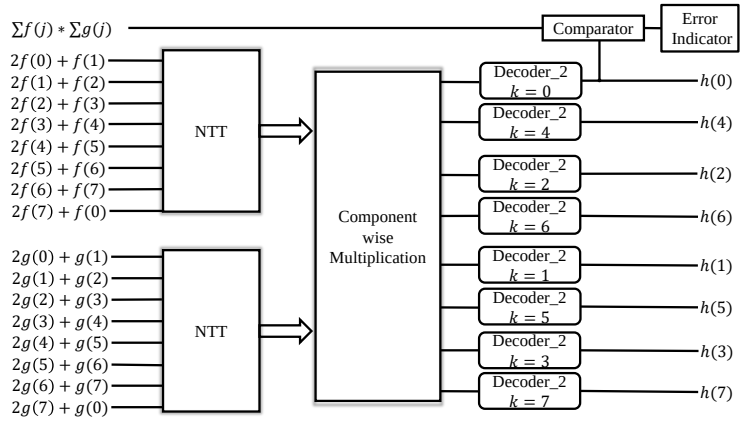

Fig. 2. Proposed algorithm level error detection scheme for the NTT multiplication module using NWC.

where α and β in Equation (8) are scalars which can be selected arbitrarily. Let us denote $\tilde{f}^m$ and $\tilde{g}^m$, respectively, as the inputs which are shifted by m. In other words, we can achieve:

$$\hat{f} = \frac{1}{(\alpha + \beta\omega^{-k})} NTT(\alpha\tilde{f} + \beta\tilde{f}^1), \tag{9}$$

$$\hat{g} = \frac{1}{(\alpha + \beta\omega^{-k})} NTT(\alpha\tilde{g} + \beta\tilde{g}^1). \tag{10}$$

Let us denote h as the component-wise NTT multiplication of $\tilde{f}$ and $\tilde{g}$. We can get:

$$h = NTT(\tilde{f}) \circ NTT(\tilde{g}) = \frac{1}{(\alpha + \beta\omega^{-k})^2} NTT(\alpha\tilde{f} + \beta\tilde{f}^1) \circ NTT(\alpha\tilde{g} + \beta\tilde{g}^1). \tag{11}$$

The proposed error detection scheme is depicted in Figure 2. The Decoder_2 sub-block multiplies its input with $\frac{1}{(\alpha+\beta\omega^{-k})^2}$, which is the inverse of $(\alpha + \beta\omega^{-k})^2$ in $\mathbb{R} = \mathbb{Z}[X]/(X^n + 1)$. The scheme for error detection involves comparing $h(0)$, obtained from (11), with $\sum_{j=0}^{n-1} \tilde{f}(j) \sum_{j=0}^{n-1} \tilde{g}(j)$ obtained from Equation (7).

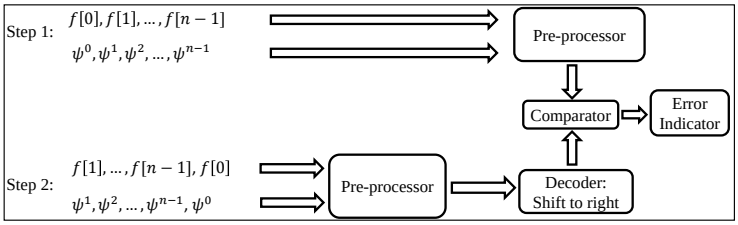

Fig. 3. Proposed error detection scheme for pre-process module through recomputation with shifted operands.

3.1.2 Pre-Processor. The pre-processor module performs element-by-element multiplication on two input arrays. In proposing the error detection scheme, we have applied an additional recomputing with shifted operands. As shown in Figure 3, the encoding and decoding modules constitute shifting which is free in hardware. We emphasize that our error detection mechanism is designed to monitor the correctness of pre-processing, independent of when it is computed. As such, our countermeasure remains applicable even when these operations are merged into the butterfly units.

3.2 Kyber

To apply the NWC, it is necessary that $2N \mid (q-1)$. In other words, $q-1$ must be multiple of $2N$ to apply NWC. For the Kyber parameters with the prime $q = 3329$ and $N = 256$, the base field $\mathbb{Z}_q$ includes 256-th roots of unity but lacks 512-th roots. Therefore, NWC cannot be used in Kyber.

3.2.1 NTT Module. By expressing Equations (2) and (3) as a matrix multiplication, we obtain the following relation where $N = 256$ with respect to Kyber parameters:

$$f'_{even} = \begin{bmatrix} \hat{f}(0) \\ \hat{f}(2) \\ . \\ \hat{f}(252) \\ \hat{f}(N-2) \end{bmatrix} = \partial * f_{even}, \text{ and } f'_{odd} = \begin{bmatrix} \hat{f}(1) \\ \hat{f}(3) \\ . \\ \hat{f}(253) \\ \hat{f}(N-1) \end{bmatrix} = \partial * f_{odd}, \text{ where } f_{even} = \begin{bmatrix} f(0) \\ f(2) \\ . \\ f(252)) \\ f(N-2) \end{bmatrix},$$

$$f_{odd} = \begin{bmatrix} f(1) \\ f(3) \\ . \\ f(253) \\ f(N-1) \end{bmatrix}, \text{ and } \partial = \begin{bmatrix} 1 & \omega & \omega^2 & \ldots & \omega^{127} \\ 1 & \omega^{2b(1)+1} & \omega^{2(2b(1)+1)} & \ldots & \omega^{127(2b(1)+1)} \\ 1 & \omega^{2b(2)+1} & \omega^{2(2b(2)+1)} & \ldots & \omega^{127(2b(2)+1)} \\ . & . & . & \ldots & . \\ 1 & \omega^{2b(127)+1} & \omega^{2(2b(127)+1)} & \ldots & \omega^{127(2b(127)+1)} \end{bmatrix}.$$

If we rotate the input sequence of f'_{even} and f'_{odd} by one, by using the symmetric and periodicity properties ($\omega^{k+N/2} = -\omega^k$ and $\omega^{k+N} = \omega^k$) of NTT, we get:

$$f''_{even} = \begin{bmatrix} \frac{\hat{f}(0)-2f(0)}{\omega^{2b(0)+1}} \\ \frac{\hat{f}(2)-2f(0)}{\omega^{2b(1)+1}} \\ . \\ \frac{\hat{f}(252)-2f(0)}{\omega^{2b(126)+1}} \\ \frac{\hat{f}(N-2)-2f(0)}{\omega^{2b(\frac{N-2}{2})+1}} \end{bmatrix} = \partial * \begin{bmatrix} f(2) \\ f(4) \\ . \\ f(N-2) \\ f(0) \end{bmatrix}, \tag{12}$$

$$f''_{odd} = \begin{bmatrix} \frac{\hat{f}(1)-2f(1)}{\omega^{2b(0)+1}} \\ \frac{\hat{f}(3)-2f(1)}{\omega^{2b(1)+1}} \\ . \\ \frac{\hat{f}(253)-2f(1)}{\omega^{2b(126)+1}} \\ \frac{\hat{f}(N-1)-2f(1)}{\omega^{2b(\frac{N-2}{2})+1}} \end{bmatrix} = \partial * \begin{bmatrix} f(3) \\ f(5) \\ . \\ f(N-1) \\ f(1) \end{bmatrix}. \tag{13}$$

From $\alpha f'_{even} + \beta f''_{even}$ and $\alpha f'_{odd} + \beta f''_{odd}$, we get the following:

$$\begin{bmatrix} \alpha\hat{f}(0) + \frac{\beta(\hat{f}(0)-2f(0))}{\omega^{2b(0)+1}} \\ \alpha\hat{f}(2) + \frac{\beta(\hat{f}(2)-2f(0))}{\omega^{2b(1)+1}} \\ . \\ \alpha\hat{f}(252) + \frac{\beta(\hat{f}(252)-2f(0))}{\omega^{2b(126)+1}} \\ \alpha\hat{f}(N-2) + \frac{\beta(\hat{f}(N-2)-2f(0))}{\omega^{2b(127)+1}} \end{bmatrix} = \partial * \begin{bmatrix} \alpha f(0) + \beta f(2) \\ \alpha f(2) + \beta f(4) \\ . \\ \alpha f(252) + \beta f(N-2) \\ \alpha f(N-2) + \beta f(0) \end{bmatrix}, \tag{14}$$

$$\begin{bmatrix} \alpha\hat{f}(1) + \frac{\beta(\hat{f}(1)-2f(1))}{\omega^{2b(0)+1}} \\ \alpha\hat{f}(3) + \frac{\beta(\hat{f}(3)-2f(1))}{\omega^{2b(1)+1}} \\ . \\ \alpha\hat{f}(253) + \frac{\beta(\hat{f}(253)-2f(1))}{\omega^{2b(126)+1}} \\ \alpha\hat{f}(N-1) + \frac{\beta(\hat{f}(N-1)-2f(1))}{\omega^{2b(127)+1}} \end{bmatrix} = \partial * \begin{bmatrix} \alpha f(1) + \beta f(3) \\ \alpha f(3) + \beta f(5) \\ . \\ \alpha f(253) + \beta f(N-1) \\ \alpha f(N-1) + \beta f(1) \end{bmatrix}. \tag{15}$$

Let us denote f^1_{even} and f^1_{odd}, respectively, as the f_{even} and f_{odd} that are shifted by 1. If we call Equations (2) and (3) NTT_Kyber, we can achieve:

$$\hat{f}(2k) = \frac{1}{(\frac{\beta}{\omega^{2b(k)+1}} + \alpha)} \left(\text{NTT\_Kyber}(\alpha f(2k) + \beta f(2k)^1) + \frac{2\beta f(0)}{\omega^{2b(k)+1}} \right), \quad k = 0, 1, \ldots, 127, \tag{16}$$

$$\hat{f}(2k+1) = \frac{1}{(\frac{\beta}{\omega^{2b(k)+1}} + \alpha)} \left(\text{NTT\_Kyber}(\alpha f(2k+1) + \beta f(2k+1)^1) + \frac{2\beta f(1)}{\omega^{2b(k)+1}} \right), \quad k = 0, 1, \ldots, 127. \tag{17}$$

We can use the relation presented below for our error detection scheme:

$$128(f(0) + f(1)) \bmod q = \sum_{j=0}^{n-1} \hat{f}(j) \bmod q, \quad q = 3329. \tag{18}$$

The proposed error detection scheme is depicted in Figure 4. The Decoder sub-block adds its input with $\frac{2\beta f(1)}{\omega^{2b(k)+1}}$ or $\frac{2\beta f(0)}{\omega^{2b(k)+1}}$ based on its index and then multiplies it with $\frac{1}{(\frac{\beta}{\omega^{2b(k)+1}}+\alpha)}$, which is the inverse of $(\frac{\beta}{\omega^{2b(k)+1}} + \alpha)$ in $\mathbb{R} = \mathbb{Z}[X]/(X^n + 1)$. The scheme for error detection involves comparing $128(f(0) + f(1)) \bmod q$, with $\sum_{j=0}^{n-1} \hat{f}(j) \bmod q$.

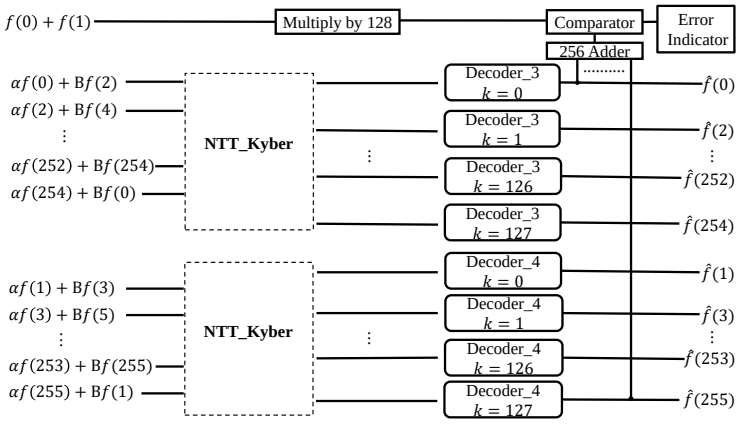

Fig. 4. Proposed algorithm level error detection scheme for the NTT utilized in Kyber.

ALGORITHM 5: CPA Kyber KeyGen and Encrypt Functions

KeyGen():
  $\rho, \sigma \leftarrow \mathcal{U}(\{0,1\}^{256})$
  $A \leftarrow \text{GenMatrix}(\rho)$
  $(s, e) \leftarrow \text{SampleNoise}(\sigma)$
  $s' \leftarrow \text{NTT}(s)$
  $e' \leftarrow \text{NTT}(e)$
  $t \leftarrow A \circ s' + e'$
  $pk \leftarrow (\text{Encode}(t), \rho)$
  $sk \leftarrow \text{Encode}(s')$
  return (pk, sk)
Encrypt$(pk(t, \rho), m, r)$:
  $(t, \rho) \leftarrow pk$
  $r \leftarrow \text{SampleNoise}(r)$
  $e_1, e_2 \leftarrow \text{SampleNoise}(r)$
  $r' \leftarrow \text{NTT}(r)$
  $u \leftarrow \text{INTT}(A^T \circ r') + e_1$
  $v \leftarrow \text{INTT}(t^T \circ r') + e_2 + \text{Encode}(m)$
  $c_1 \leftarrow \text{Encode}(u)$
  $c_2 \leftarrow \text{Encode}(v)$
  return $c = (c_1, c_2)$

4 Fault Attack on Kyber

In this section, we describe [46] the first fault attack targeting the NTT module within the Kyber cryptographic scheme. We describe how our proposed error detection scheme addresses and mitigates both this fault attack and naturally occurring faults. By zeroing all twiddle factors used in the NTT operation, particularly when applied to secrets or errors, the entropy of these secrets or errors can be significantly reduced. This leads to generating faulty but valid LWE instances, making it straightforward to compromise the secret key. For instance, if the input of the NTT block is $f = (f_0, f_1, \ldots, f_{n-1})$ the faulty output after zeroing all the twiddle factors will become $\hat{f} = (f_0, 0, \ldots, 0)$. Therefore, the injected fault significantly reduces the entropy of f down to a single

Table 1. Comparison of Existing Countermeasures against Fault Attack on NTT Studies

Methods	Performance	Detect Natural Faults	Hardware Implementation	Note
Our	+	+	+	Assessed 16% overhead in clock cycles in Cortex-A72 for Kyber. Assessed 12% overhead in latency in FPGA for NWC.
Recomputation [50]	-	+	+	The overhead in area or latency is doubled based on the design.
Checking Entropy of NTT Output [46]	+	-	-	Checks on the distribution of the NTT outputs could help detect fault attack.
Polynomial Evaluation and Interpolation [5]	-	+	-	Requires 2n-1 multiplications and 2n-2 additions. Assessed 74% overhead in clock cycles in Cortex-M4.
On-the-fly Computation of Twiddle Factors [46]	-	-	-	Compute twiddle factors on-the-fly instead of precomputing. However, we lose the benefits of precomputing.

coefficient f_0, making it easily guessable by an attacker due to the limited range of secrets and errors employed in Kyber KEM. In this context, an attacker can exploit the NTT operation performed on the secret s or error e during the key-generation process (Lines 5 and 6 in Algorithm 5). Such an attack leads to the use of low-entropy secrets and errors when generating a faulty public key, enabling easy recovery of the secret key. Furthermore, an attacker can target the NTT operation within Kyber's encryption function (Line 17 in Algorithm 5), reducing the entropy of r and consequently facilitating message recovery.

4.1 Effectiveness against Fault Attack

Our error detection scheme can detect the mentioned fault attack. By focusing on Equation (7), if the fault attack happens in our error detection scheme and the input is $f = (f_0, f_1, \ldots, f_{n-1})$, the output of the decoder modules will become $\hat{f} = (\alpha f_0 + \beta f_1, 0, \ldots, 0)$. Using the proposed error detection $nf(0) \neq (\alpha f_0 + \beta f_1 = \sum_{i=0}^{n-1} \hat{f})$. Therefore, the fault is detected. It is worth noting that alternative methods to detect this particular fault attack have been discussed in Reference [46], but these methods are limited in terms of efficiency and their ability to detect natural faults. Table 1 describes various countermeasures against the explained fault attack.

4.2 Hardware/Software Fault Attacks Techniques

Software-based attacks such as Rowhammer [28] enable remote attackers to deliberately alter bits in a target process's memory, potentially triggering unexpected or malicious outcomes. The authors in [3] showed Rowhammer based attacks on Kyber targeting the NTT module. The authors recommended countermeasures such as limiting the amount of time that key material is in a vulnerable memory location or using vectorized instruction to remove the nested loop in the NTT reference implementation in Kyber. In hardware-based fault attacks, adversaries typically rely on external stimuli such as **electromagnetic** (**EM**) pulses, laser shots, or clock/voltage glitches to alter device behavior at runtime. The mentioned fault attack on Kyber [46] focused on NTT, demonstrates a practical example of this by using EM fault injection on an ARM Cortex-M4 microcontroller running optimized Kyber and Dilithium code from the pqm4 library. By precisely timing the injection, the authors managed to zeroize all twiddle constants loaded for the NTT operation using only a single, targeted fault. This drastically reduces the output's entropy. The work shows how even small, well-timed hardware perturbations can undermine cryptographic primitives at the arithmetic level, and emphasize that safeguards against such physical injections must be carefully designed to preserve security in real-world settings.

ARCHIE [20] and FIVER [48] can be employed for injecting faults in software and hardware implementations, respectively. These platforms provide a standardized and comprehensive fault injection framework, enabling an effective assessment of the robustness of proposed countermeasures.

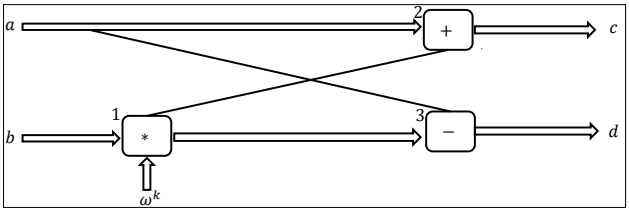

Fig. 5. The utilized fault model in this work for the butterfly sub-block.

5 Error Coverage Simulation Results

5.1 Fault Model

The butterfly module is a key element in the NTT that enables efficient modular transformations, much like its function in the FFT. It processes two input elements by calculating their sum and difference. Its ability to handle modular arithmetic and root of unity multiplications is essential for enhancing the performance of the NTT.

As depicted in Figure 5, faults can happen at modules {1,2, or 3} marked in the figure, which may result in erroneous outputs of a butterfly sub-block used in the NTT module. Fault occurrence follows a normal distribution within butterfly modules in Figures 2 and 4. Furthermore, as we provide fault detection schemes over the NTT multiplication, faults can happen during component wise multiplication module. The NTT process involves numerous butterfly operations, where faults may occur at any of these operations. Within each butterfly, faults can happen at positions {1, 2, or 3}. The fault attack we discussed in Section 4 is based on zeroing the twiddle factors. As illustrated in Figure 5, any fault injected into the twiddle factors can indeed be effectively modeled as a fault in the multiplication module (Box 1). In our fault model, no two faults are allowed to occur in the same butterfly operation. In the normal mode fault injection method, if the NTT process contains N butterfly operations and F represents the total number of faults, each butterfly operation has a probability of $\frac{F}{N}$ of being faulty. Moreover, if a butterfly is faulty, the probability of the fault occurring at position {1, 2, or 3} is $\frac{1}{3}$.

In the burst mode fault injection method, once a faulty butterfly operation is randomly chosen, all subsequent butterfly operations will also be faulty. Our simulation and implementation code is publicly available in our github.[1]

5.2 Fault Simulation

To assess the error coverage, we employed Python3 to perform simulations, applying the described fault models and error detection techniques to both the NTT and pre-process modules.

5.2.1 `Negative Wrapped Convolution`. The simulations encompassed a million samples, using parameters identical to the standard Kyber Round 1. As shown in Table 2, with increase in the number of faults, we can attain an error detection ratio close to 100% for both modules. Higher error coverage is achievable for identifying burst errors.

The table provides data comparing the effectiveness of the proposed fault injection on two cryptographic components: the Pre-process stage and the NTT Multiplication stage, based on the number of faults injected (1, 2, 4, 8, or 16). For the Pre-process component, for a single fault, we already get high error coverage of 99.7%. As the number of injected faults increases, the ratio quickly reaches 100%. In contrast, the NTT Multiplication stage is more difficult to protect, but

[1] https://github.com/KasraAhmadi/NTT_Error_Detection

Table 2. NWC Achieved Error Detection Ratios of the Proposed Schemes with 1, 2, 4, 8, and 16 Faults Occurrence with Normal Fault Injection Method for One Million Samples

Parameters		[1]$n = 256$,[2] $\omega = 3,844$, [3]$q = 7,681$	
Component		**Pre-process**	**NTT Multiplication**
Number of faults	1	99.7%	53%
	2	99.9%	70.6%
	4	100%	90.9%
	8	100%	99%
	16	100%	99.9%

[1]n = Polynomial size, [2]ω = Twiddle factor, [3]q = Prime number.

eventually we get to close to 100%. With one injected fault, 53% is the detection ratio. However, the success rate increases as more faults are injected: 70.6% for 2 faults, 90.9% for 4, 99% for 8, and 99.9% for 16 faults.

5.2.2 Kyber. In the standard Kyber Round 3 NTT implementation, there are 896 butterfly operations, corresponding to the polynomial size of 256. A fault can potentially occur at any of these 896 butterfly operations. The simulations involved one million samples, using the same parameters as in Kyber Round 3.

As indicated in Table 3, increasing the number of faults leads to an error detection ratio approaching 100% in both normal and burst modes. In Kyber's NTT implementation, the error detection ratios for various fault occurrences were analyzed using one million samples, with parameters set to $n = 256$, $\omega = 17$, and $q = 3329$. The results show that as the number of injected faults increases, the error detection ratio improves significantly. The trend of error detection ratios in Table 3 highlights the robustness of the error detection mechanism in Kyber's NTT when subjected to increasing numbers of faults. A higher number of faults leads to near-perfect or perfect error detection, suggesting the scheme becomes more effective at fault identification as fault intensity increases. The NTT in Kyber was also tested using a burst fault model, with error detection ratios evaluated over one million samples. In the burst fault model, multiple consecutive bits in the data are corrupted simultaneously. The error detection ratios trend in Table 3 shows that as the number of burst faults increases, the error detection capability of the scheme improves, approaching almost 100% accuracy with 5 or more faults. This indicates the robustness of the detection method, particularly when dealing with higher levels of fault occurrences.

6 Hardware Design

To demonstrate the low area and low latency overhead of our error detection scheme in FPGA implementation, we built upon the approach presented in Reference [36], an extensive open-source study on flexible NTT design methodologies, and integrated our proposed error detection techniques into this design. This work introduces an adaptable yet efficient hardware generator, which we used as our baseline implementation. Furthermore, the unified NTT architecture proposed in our design shares several similarities with the KiD framework introduced by Reference [35], particularly in its use of configurable radix-2 butterfly units, shared memory architecture, and pipelined dual-port BRAM organization. Their approach demonstrates an efficient unified implementation for both Kyber and Dilithium on FPGAs, which aligns with our motivation to support resource-constrained post-quantum cryptographic applications using compact and high-performance NTT

Table 3. NTT in Kyber Achieved Error Detection Ratios of the Proposed Schemes with Normal and Burst Fault Injection Methods for One Million Samples

Parameters		$n = 256, \omega = 17$ $q = 3,329$		
Fault injection method		Normal mode	Burst mode	
Number of faults	1	74.9%	2	93.82%
	2	93.45%	3	98.3%
	4	99.49%	4	99.42%
	8	99.95%	5	99.76%
	16	100%	6	99.8%

cores. We then integrated the necessary modules for encoding and decoding to support both error detection schemes designed for `NWC` and `Kyber`. The design employs the Gentleman-Sande butterfly configuration, where the butterfly unit receives input in normal order and produces output in bit-reversed order. The NTT operation can be executed in parallel by carrying out multiple butterfly operations simultaneously. Each **Processing Element** (**PE**) contains a butterfly unit that performs arithmetic computations, including modular addition, subtraction, and multiplication.

We conducted a benchmark for implementation on FPGA, i.e., Xilinx/AMD Artix-7. The Xilinx Artix-7 is a lower-cost, mid-range FPGA aimed at applications that need efficient power usage and moderate performance, such as communications, video processing, and embedded systems. It does not include integrated processors, relying solely on programmable logic. The results clearly demonstrate that our proposed efficient error detection schemes maintain a modest overhead while effectively achieving a high level of fault detection.

6.1 Montgomery Modular Multiplication

The modular multiplier is a crucial element of PE, comprising two main components: An integer multiplier followed by a modular reduction module. For modular reduction, Montgomery reduction is used extending the word-level Montgomery modular reduction algorithm to accommodate NTT-friendly primes [37]. Each multiplier input is partitioned into 16-bit segments, with each 16-bit × 16-bit multiplication handled by a dedicated DSP block. The intermediate results are then accumulated using carry-save adders to compute the final product. Additionally, the proposed integer multiplier is fully pipelined, enabling it to generate one multiplication result per clock cycle.

The Montgomery reduction algorithm takes $C = A * B$ as input and computes the result $A * B * R^{-1} \pmod q$ where R is the Montgomery reduction residual. To obtain $C = A * B \pmod q$, the residual must be adjusted with an additional multiplication by R. However, this extra step can be moved to the input stage by pre-multiplying of the inputs by R. In the PE module, where one of the inputs is the fixed twiddle factor ω, this extra multiplication can be fused by pre-computing $\omega * R \pmod q$ and use instead of previous pre-computed twiddle factors.

6.2 PE and Butterfly Unit

Each PE is designed using the Gentleman-Sande butterfly structure, as shown in Figure 6. A PE takes in two coefficients and a twiddle factor, performs the butterfly computation, and outputs two results: The even (E) and odd (O) coefficients. To carry out this operation, each PE includes a modular addition, a modular subtraction, and a Montgomery modular multiplication. Additionally, each PE uses three dualport BRAMs, two are dedicated to storing input and intermediate coefficients, while the third, known as the TW BRAM, holds precomputed twiddle factors. The mode and auxdata

inputs are integrated into the original PE design to enable reconfiguration of the PE, allowing it to support the decoder module used in error detection.

6.3 Area Optimization of Proposed Error Detection Schemes

The proposed error detection scheme utilizes an encoder module, a decoder module, and an error indicator module. The encoder and error indicator modules are lightweight since their implementation relies on addition and shifting. However, the decoder module is comparatively heavier due to the use of multiplication. Therefore, to minimize the area overhead caused by multiplication, we propose utilizing the existing PE modules in the baseline implementation.

As shown in Figure 6, setting the mode to zero configures the PE for forward NTT computation, while setting it to one enables it to function as a Montgomery multiplication unit, performing the operation $O = auxdata * decparam \pmod q$. The proposed decoder module in `Kyber` requires two multiplications and one addition, while in `NWC` it relies on a single multiplication, we propose reusing the PE modular multiplication unit for decoding to achieve a more efficient and compact design.

To achieve fully pipelined decoder modules and efficiently reuse the existing PEs for modular multiplication, the number of PEs in both designs must be at least equal to the number of multiplications in the decoder modules. As a result, the baseline NTT implementation for `Kyber` utilizes two PEs, while the `NWC` is implemented with a single PE.

6.4 `Kyber`

6.4.1 Encoder. Based on Equations (16) and (17), our encoder module is $\alpha f(k) + \beta f(k)^1$. If we choose $\alpha = 1$ and $\beta = 2$, as used in our simulation results, the proposed encoder module relies on shifting that is free in hardware and a single addition.

6.4.2 Decoder. Since multiplication consumes the most area due to DSP usage, we leverage the existing multiplication module in the PE to optimize area efficiency in our error detection design. The decoder module depicted in Figure 7 performs the following three steps for $k \in \{1, 2, \dots, 255\}$:

$$Step_1 : \quad out_1(k) = \begin{cases} f(0) \times \frac{2\beta}{\omega^{2b(k/2)+1}}, & \text{if } k \bmod 2 = 0 \\ f(1) \times \frac{2\beta}{\omega^{2b((k-1)/2)+1}}, & \text{if } k \bmod 2 = 1 \end{cases}$$

$$Step_2 : \quad out_2(k) = out_1(k) + Encoded_{data}(k)$$

$$Step_3 : \quad out_3(k) = \begin{cases} out_2(k) \times \frac{1}{(\frac{\beta}{\omega^{2b(k/2)+1}} + \alpha)}, & \text{if } k \bmod 2 = 0 \\ out_2(k) \times \frac{1}{(\frac{\beta}{\omega^{2b((k-1)/2))+1}} + \alpha)}, & \text{if } k \bmod 2 = 1 \end{cases}$$

We implemented pipelining in the decoder module to produce result every clock cycle. By precomputing and saving $\frac{2\beta}{\omega^{2b(k)+1}}$ in BRAM Decoder Factor_1 and $\frac{1}{(\frac{\beta}{\omega^{2b(k)+1}} + \alpha)}$ in BRAM Decoder Factor_2, the decoder module is based on two multiplications and one addition.

6.4.3 Error Detection Module. Figure 8 depicts Equation (18) as implemented in our hardware design. The error detection module utilizes modular addition, similar to that used in the PE, along with XOR and shift operations.

6.5 `Negative Wrapped Convolution`

We focused on Component-wise NTT Multiplication for the hardware implementation. As discussed in Section 6.3, we utilized a single PE in our hardware design. The encoder module used in `NWC` is

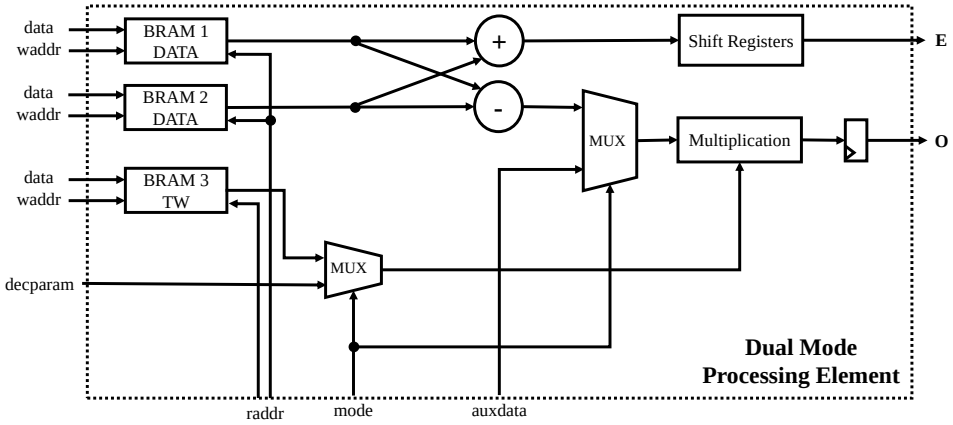

Fig. 6. Dual mode PE and butterfly unit.

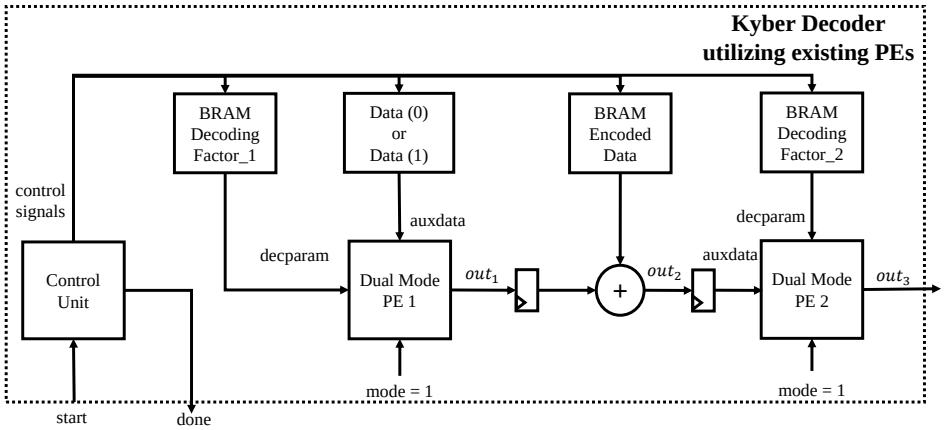

Fig. 7. Decoder module used in Kyber.

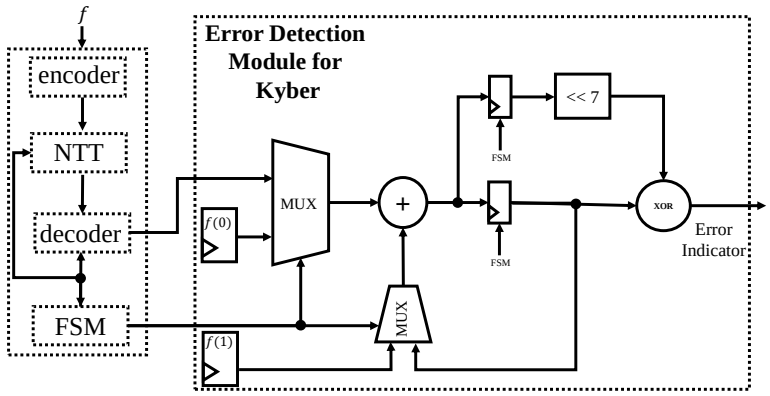

Fig. 8. Error detection module for Kyber.

Table 4. AMD/Xilinx Artix-7, xc7a200tfbg676-2 Hardware Implementation Results

Strategy		**Kyber**		**NWC**	
Scheme		Our Scheme	Baseline Work [36]	Our Scheme	Baseline Work [36]
PE		2	2	1	1
Area	LUTs	1,020	971	718	661
	DSPs	6	6	3	3
	BRAMs	6	4	4.5	3.5
Timing	Latency [CCs]	920	648	2,378	2,112
	Total time [ns]	6,568	4,626	16,978	15,079
Power (W) @ 140 MHz		0.16	0.15	0.17	0.16
Energy (nJ)		1,050	693	2,886	2,412

Baseline work: The design which does not include any error detection.
Our Scheme: The design which includes error detection scheme.

the same one as used in Kyber. The decoder module is derived from Equation (11), which involves a single multiplication. To optimize area usage, we reuse the existing PE module in mode 1 to perform this multiplication. The error detection module is based on Equation (7). $\sum_{j=0}^{n-1} \tilde{f}(j) \sum_{j=0}^{n-1} \tilde{g}$ is independent from NTT module and can be computed independently in $2n + 5$ cycles using single modular addition module and single PE module. $\hat{f}(0)\hat{g}(0)$ is computed using existing PE module in NTT with mode 1. From an area perspective, the error detection framework relies on a single PE module (shared with the NTT module), one modular addition unit, and an XOR module.

Table 4 presents the FPGA implementation results for both NWC and Kyber schemes. It compares our proposed design, which incorporates fault detection, against a baseline design on the AMD/Xilinx Artix-7 FPGA (xc7a200tfbg676-2), focusing on resource usage, power consumption, and performance metrics. For Kyber, our design consumes more hardware resources than the baseline. It utilizes more LUTs (1,020 vs. 971), more BRAMs (6 vs. 4), while both use the same number of DSPs (6). Power usage is comparable, with our design drawing slightly more power (0.16W vs. 0.15W). The latency of our design is higher (920 vs. 648 clock cycles), which is expected due to the added fault detection capability absent in the baseline. Similarly, for NWC, our design implementation shows increased usage of LUTs (718 vs. 661), BRAMs (4.5 vs. 3.5), and maintains the same number of DSPs (3). Power consumption remains close, with our design at 0.17W compared with 0.16W for the baseline. Again, the latency is slightly higher (2,378 vs. 2,112 clock cycles) due to the integrated error detection features. Both implementations of our scheme maintain a maximum operating frequency of approximately 140 MHz across the tested FPGAs.

7 HLS and Processor Based Design

7.1 Negative Wrapped Convolution

We utilized the **High-Level Synthesis** (**HLS**) Vitis development environment of AMD/Xilinx to transform our proposed schemes into **Register Transfer Level** (**RTL**) hardware descriptions. The generated IP imported to Vivado to report power, utilization, and latency. Table 5 demonstrates the results of our HLS implementations and the derivations for area, timing, power, and energy for NWC. In the Vivado synthesis tool context, area is determined by combining Slices and DSPs, with a conversion ratio where one DSP is deemed equivalent to 100 Slices. Our proposed error detection

Table 5. AMD/Xilinx Artix-7 xc7a75ti-ftg256-1L NWC HLS Implementation Results

Strategy		**Area Effort**		**Timing Effort**	
Scheme		Our scheme	Baseline work	Our scheme	Baseline work
Area	LUTs	1,569	1,530	2,956	2,939
	FFs	1,693	1,597	2,979	2,883
	SLICEs	673	656	1,240	1,253
	DSPs	33	30	54	51
Power (W) @ 140 MHz		0.2	0.19	0.28	0.26
Timing	Latency [CCs]	3,749	3,482	2,267	2,003
	Total time [ns]	26,767	24,861	16,186	14,301
Energy (nJ)		5,353	4,723	4,532	3,718

scheme imposes a maximum overhead of 9% in additional area and introduces a latency increase of up to 13% in clock cycles, at most considering different designs.

7.1.1 HLS Optimization. The HLS-based implementation does not outperform the FPGA design, despite significant effort in optimizing the algorithm and fine-tuning HLS directives. A key observation is the substantial difference in DSP usage, while the FPGA implementation maintains efficient resource utilization, the HLS version consumes considerably more DSP blocks, highlighting the limitations of automated resource mapping. Nonetheless, HLS offers a significantly faster development cycle, making it attractive for rapid prototyping and design exploration. However, when targeting performance, critical applications, traditional RTL-based FPGA implementation provides better control over resource allocation and timing, leading to superior area and performance efficiency.

Although HLS has shifted the design entry level of abstraction from RTL to C/C++, practical implementation often requires significant source code rewriting to make it HLS-ready. We have carefully taken this into effort as discussed here. This includes the incorporation of pragmas to attain satisfactory performance. Our intended area and timing efforts implementation was realized by strategically inserting the following pragmas into our program.

7.1.2 Pre-Calculate the Decoder Module Values. To enhance the efficiency of the proposed error detection scheme, we have the option to pre-calculate the decoder module in Equation (11), specifically computing $\frac{1}{(\alpha+\beta\omega^{-k})^2}$ for various values of the parameter k and save them in memory.

7.1.3 Utilizing Task-Level Pipelining. We perform loop unrolling on the outer loop in Algorithm 1 (Line 2) and transform the remaining code into a function named "stage," illustrating each stage in the NTT architecture. We undertook this approach to leverage task-level pipelining, allowing functions and loops to operate simultaneously. This enhances the concurrency of the RTL implementation, resulting in an overall increase in design throughput.

Table 6. Software Implementation Results for the NTT in Kyber

	ARM v8 Cortex-A72 @ 1.5GHz	Intel Core-i7 @ 2.4GHz
Baseline work	8,859 clock cycles	5,814 clock cycles
Our Scheme	10,316 clock cycles	7,443 clock cycles

7.1.4 Pipelining the Stage Function. This pragma was inserted to facilitate instruction-level pipelining, aiming to enhance throughput and clock frequency within each NTT stage function. It is important to acknowledge that this optimization entails the tradeoff of utilizing extra digital resources.

7.2 Kyber

We implemented our proposed error detection scheme for the official Kyber NIST submission (Round 3) on both x86-64 architecture and a 12th Gen Intel Core i7-12800H 2.4GHz processor, as well as on more resource-constrained devices, such as the 2.4GHz quad-core ARM Cortex-A72 (Raspberry Pi). We utilized the PAPI library to evaluate the performance of the software implementation and measure the overhead introduced by our error detection scheme. The PAPI library provides a standardized interface for accessing performance counters in CPUs, enabling researchers/developers to measure and analyze the performance of applications efficiently. Table 6 demonstrates the results of our implementation using standard Kyber Round 3 implementation as baseline work on 2 different architectures. The code for our simulation and implementation is accessible on our GitHub. On the ARM Cortex-A72, the baseline work requires 8,859 clock cycles to execute the NTT operation in Kyber Round 3, whereas our scheme takes 10,316 clock cycles, an increase of approximately 16.5%. This suggests that the additional error detection or enhancements come with a performance cost in terms of clock cycles.

Similarly, on the Intel Core-i7, the baseline work completes in 5,814 clock cycles, while our scheme takes 7,443 clock cycles, an increase of about 28%. The performance overhead on the Intel Core-i7 is higher compared with the ARM v8, indicating that the performance impact of the error detection or other improvements is more significant on this processor.

7.2.1 Implementation Optimization. According to Equations (16) and (17), the overhead introduced by our error detection scheme depends on the encoding and decoding modules. The encoding module involves one addition and one shift operation. For the decoder, we can store coefficients $(\frac{1}{(\frac{\beta}{\omega^{2b(k)+1}}+\alpha)}$ and $\frac{2\beta}{\omega^{2b(k)+1}})$ in memory to eliminate additional computations. The decoding module comprises two multiplications and one addition.

8 Conclusion

In this article, we proposed efficient algorithm level error detection schemes over the NTT operation used in `NWC` and `Kyber`. We have aimed at achieving low hardware overhead and low latency, suitable for deeply-embedded systems. The presented scheme effectively safeguards cryptographic algorithms that employ the NTT multiplication. By performing simulations, we demonstrated that the proposed error detection schemes achieved extensive coverage of faults. Moreover, we implemented our proposed error detection schemes on Xilinx/AMD FPGAs, HLS, and processor-based architectures. Regarding overhead and latency, the implementation led to minimal additional expenses in hardware and software. With high error coverage and acceptable overhead, the proposed schemes in this work are suitable for resource-constrained and sensitive usage models.

References

[1] Kasra Ahmadi, Saeed Aghapour, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2024. Efficient error detection cryptographic architectures benchmarked on FPGAs for montgomery ladder. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2024), 1–0. DOI: https://doi.org/10.1109/TVLSI.2024.3419700

[2] Kasra Ahmadi, Saeed Aghapour, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2024. Efficient error detection schemes for ECSM window method benchmarked on FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 32, 3 (2024), 592–596. DOI: https://doi.org/10.1109/TVLSI.2023.3341147

[3] Samy Amer, Yingchen Wang, Hunter Kippen, Thinh Dang, Daniel Genkin, Andrew Kwong, Alexander Nelson, and Arkady Yerukhimovich. 2025. PQ-Hammer: End-to-end key recovery attacks on post-quantum cryptography using rowhammer . In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 48–48. DOI: https://doi.org/10.1109/SP61157.2025.00048

[4] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2019. CRYSTALS-kyber algorithm specifications and supporting documentation. *NIST PQC Round* 2, 4 (2019), 1–43.

[5] Sven Bauer, Fabrizio De Santis, Kristjane Koleci, and Anita Aghaie. 2024. A fault-resistant NTT by polynomial evaluation and interpolation. *Cryptology ePrint Archive* (2024).

[6] Nina Bindel, Johannes Buchmann, and Juliane Krämer. 2016. Lattice-based signature schemes and their sensitivity to fault attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 63–77.

[7] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. 2021. High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography. In *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*. IEEE, 94–101.

[8] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 353–367.

[9] Leon Groot Bruinderink and Peter Pessl. 2018. Differential fault attacks on deterministic lattice signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 3 (2018), 21–43.

[10] Alvaro Cintas Canto, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2022. Reliable constructions for the key generator of code-based post-quantum cryptosystems on FPGA. *ACM Journal on Emerging Technologies in Computing Systems* 19, 1 (2022), 1–20.

[11] Alvaro Cintas Canto, Ausmita Sarker, Jasmin Kaur, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2022. Error detection schemes assessed on FPGA for multipliers in lattice-based key encapsulation mechanisms in post-quantum cryptography. *IEEE Transactions on Emerging Topics in Computing* 11, 3 (2022), 791–797.

[12] William T. Cochran, James W. Cooley, David L. Favin, Howard D. Helms, Reginald A. Kaenel, William W. Lang, George C. Maling, David E. Nelson, Charles M. Rader, and Peter D. Welch. 1967. What is the fast fourier transform? *Proc. IEEE* 55, 10 (1967), 1664–1674.

[13] James W. Cooley and John W. Tukey. 1965. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation* 19, 90 (1965), 297–301.

[14] Jeroen Delvaux. 2021. Roulette: A diverse family of feasible fault attacks on masked kyber. *Cryptology ePrint Archive* (2021).

[15] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 1 (2018), 238–268.

[16] Mohamed ElGhamrawy, Melissa Azouaoui, Olivier Bronchain, Joost Renes, Tobias Schneider, Markus Schönauer, Okan Seker, and Christine van Vredendaal. 2023. From MLWE to RLWE: A differential fault attack on randomized & deterministic dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023, 4 (2023), 262–286.

[17] Thomas Espitau, Pierre-Alain Fouque, Benoit Gerard, and Mehdi Tibouchi. 2018. Loop-abort faults on lattice-based signature schemes and key exchange protocols. *IEEE Trans. Comput.* 67, 11 (2018), 1535–1549.

[18] Yue Geng, Xiao Hu, Minghao Li, and Zhongfeng Wang. 2023. Rethinking parallel memory access pattern in number theoretic transform design. *IEEE Transactions on Circuits and Systems II: Express Briefs* 70, 5 (2023), 1689–1693.

[19] Ying Guo, Wenfen Liu, Wen Chen, Qingwen Yan, and Yongcan Lu. 2024. ECLBC: A lightweight block cipher with error detection and correction mechanisms. *IEEE Internet of Things Journal* 11, 12 (2024), 21727–21740. DOI: https://doi.org/10.1109/JIOT.2024.3376527

[20] Florian Hauschild, Kathrin Garb, Lukas Auer, Bodo Selmke, and Johannes Obermaier. 2021. ARCHIE: A QEMU-based framework for architecture-independent evaluation of faults. In *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE, 20–30.

[21] Daniel Heinz and Thomas Pöppelmann. 2022. Combined fault and DPA protection for lattice-based cryptography. *IEEE Trans. Comput.* 72, 4 (2022), 1055–1066.

[22] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. 2021. Fault-enabled chosen-ciphertext attacks on Kyber. In *Progress in Cryptology–INDOCRYPT 2021: 22nd International Conference on Cryptology in India, Jaipur, India, December 12–15, 2021, Proceedings 22*. Springer, 311–334.

[23] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. 2023. Adapting belief propagation to counter shuffling of NTTs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023), 60–88.

[24] Vincent Hwang, Jiaxiang Liu, Gregor Seiler, Xiaomu Shi, Ming-Hsien Tsai, Bow-Yaw Wang, and Bo-Yin Yang. 2022. Verified NTT multiplications for NISTPQC KEM lattice finalists: Kyber, SABER, and NTRU. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022, 4 (2022), 718–750.

[25] Sönke Jendral. 2024. A single trace fault injection attack on hedged CRYSTALS-dilithium. *Cryptology ePrint Archive* (2024).

[26] J.-Y. Jou and Jacob A. Abraham. 1988. Fault-tolerant FFT networks. *IEEE Transactions on Computers* 37, 5 (1988), 548–561.

[27] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. 2019. pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. (2019).

[28] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 361–372.

[29] Elisabeth Krahmer, Peter Pessl, Georg Land, and Tim Güneysu. 2024. Correction fault attacks on randomized crystals-dilithium. *Cryptology ePrint Archive* (2024).

[30] Suparna Kundu, Siddhartha Chowdhury, Sayandeep Saha, Angshuman Karmakar, Debdeep Mukhopadhyay, and Ingrid Verbauwhede. 2024. Carry your fault: A fault propagation attack on side-channel protected lwe-based kem. arXiv:2401.14098. Retrieved from https://arxiv.org/abs/2401.14098

[31] Stefanus Kurniawan, Phap Duong-Ngoc, and Hanho Lee. 2023. Configurable memory-based NTT architecture for homomorphic encryption. *IEEE Transactions on Circuits and Systems II: Express Briefs* 70, 10 (2023), 3942–3946.

[32] Bin Li, Yunfei Yan, Yuanxin Wei, and Heru Han. 2023. Scalable and parallel optimization of the number theoretic transform based on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2023).

[33] Fabiano Libano, Paolo Rech, and John Brunhaver. 2023. Efficient error detection for matrix multiplication with systolic arrays on FPGAs. *IEEE Trans. Comput.* 72, 8 (2023), 2390–2403.

[34] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. 2008. SWIFFT: A modest proposal for FFT hashing. In *Fast Software Encryption: 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers 15*. Springer, 54–72.

[35] Suraj Mandal and Debapriya Basu Roy. 2024. KiD: A hardware design framework targeting unified NTT multiplication for CRYSTALS-Kyber and CRYSTALS-Dilithium on FPGA. In *2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*. IEEE, 455–460.

[36] Ahmet Can Mert, Emre Karabulut, Erdinç Öztürk, Erkay Savaş, and Aydin Aysu. 2020. An extensive study of flexible design methods for the number theoretic transform. *IEEE Trans. Comput.* 71, 11 (2020), 2829–2843.

[37] Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. 2020. Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 2 (2020), 353–362. DOI: https://doi.org/10.1109/TVLSI.2019.2943127

[38] Jianan Mu, Huajie Tan, Jiawen Wu, Haotian Lu, Chip-Hong Chang, Shuai Chen, Shengwen Liang, Jing Ye, Huawei Li, and Xiaowei Li. 2023. Energy-efficient NTT design with one-bank SRAM and 2-D PE array. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–2. DOI: https://doi.org/10.23919/DATE56975.2023.10137059

[39] Koksal Mus, Saad Islam, and Berk Sunar. 2020. QuantumHammer: A practical hybrid attack on the LUOV signature scheme. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1071–1084.

[40] NIST. 2024. (2024). Retrieved from https://csrc.nist.gov/pubs/fips/203/final, Last accessed on 2024-09-17.

[41] NIST. 2024. (2024). Retrieved from https://csrc.nist.gov/pubs/fips/204/final, Last accessed on 2024-09-17.

[42] NIST. 2024. (2024). Retrieved from https://csrc.nist.gov/pubs/fips/205/final, Last accessed on 2024-09-17.

[43] Lukas Prokop and Peter Peßl. 2021. Fault attacks on CCA-secure lattice KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 2 (2021), 37–60.

[44] Prasanna Ravi, Anupam Chattopadhyay, Jan Pieter DAnvers, and Anubhab Baksi. 2024. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (Kyber, Dilithium): Survey and new results. *ACM Transactions on Embedded Computing Systems* 23, 2 (2024), 1–54.

[45] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. 2019. Exploiting determinism in lattice-based signatures: Practical fault attacks on pqm4 implementations of NIST candidates. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 427–440.

[46] Prasanna Ravi, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. 2023. Fiddling the twiddle constants-fault injection analysis of the number theoretic transform. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023).

[47] Jan Richter-Brockmann, Pascal Sasdrich, Florian Bache, and Tim Güneysu. 2020. Concurrent error detection revisited: Hardware protection against fault and side-channel attacks. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 1–11.

[48] Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. 2021. Fiver–robust verification of countermeasures against fault injections. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 447–473.

[49] Sayandeep Saha, Manaar Alam, Arnab Bag, Debdeep Mukhopadhyay, and Pallab Dasgupta. 2023. Learn from your faults: Leakage assessment in fault attacks using deep learning. *Journal of Cryptology* 36, 3 (2023), 19.

[50] Ausmita Sarker, Alvaro Cintas Canto, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2022. Error detection architectures for hardware/software co-design approaches of number-theoretic transform. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 7 (2022), 2418–2422.

[51] Ausmita Sarker, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2022. Efficient error detection architectures for postquantum signature Falcon's sampler and KEM SABER. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 6 (2022), 794–802.

[52] Ausmita Sarker, Mehran Mozaffari-Kermani, and Reza Azarderakhsh. 2018. Hardware constructions for error detection of number-theoretic transform utilized in secure cryptographic architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 3 (2018), 738–741.

[53] Richa Singh, Saad Islam, Berk Sunar, and Patrick Schaumont. 2024. Analysis of EM fault injection on bit-sliced number theoretic transform software in dilithium. *ACM Transactions on Embedded Computing Systems* 23, 2 (2024), 1–27.

[54] Tolun Tosun and Erkay Savas. 2024. Zero-value filtering for accelerating non-profiled side-channel attack on incomplete NTT based implementations of lattice-based cryptography. *IEEE Transactions on Information Forensics and Security* (2024).

[55] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. 2021. Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part II 27*. Springer, 33–61.

[56] Cong Zhang, Dongsheng Liu, Xingjie Liu, Xuecheng Zou, Guangda Niu, Bo Liu, and Quming Jiang. 2021. Towards efficient hardware implementation of NTT for kyber on FPGAs. In *2021 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, 1–5.

[57] Neng Zhang, Qiao Qin, Hang Yuan, Chenggao Zhou, Shouyi Yin, ShaoJun Wei, and Leibo Liu. 2019. NTTU: An area-efficient low-power NTT-uncoupled architecture for NTT-based multiplication. *IEEE Trans. Comput.* 69, 4 (2019), 520–533.

Received 18 September 2024; revised 28 April 2025; accepted 11 August 2025