Hardware Constructions for Error Detection of Number-Theoretic Transform Utilized in Secure Cryptographic Architectures

Ausmita Sarker, Mehran Mozaffari-Kermani, and Reza Azarderakhsh

Abstract—Polynomial multiplication is one of the most rigorous arithmetic construction of postquantum cryptosystems. Utilizing number-theoretic transformations, the product of such multiplication can be efficiently computed in quasi-linear time $O(n \log n)$. Error detection schemes of number-theoretic transform (NTT) architectures are essential to ensure correct mathematical operations, improved security, and thwart active side-channel attacks mounted through faults. NTT is not only significant to post-quantum cryptosystems, but the structure is also valuable to the already existing security protocols, e.g., signature schemes, hash functions, and the like. This paper, for the first time, introduces new error detection schemes of NTT architectures, successfully detecting both permanent and transient faults. Our schemes are based on recomputing with negated, scaled, and swapped operands. We have implemented the proposed schemes on the application-specific integrated circuit (ASIC). Performance and implementation metrics on this hardware platform show acceptable hardware overhead. As our schemes provide acceptable complexity and high efficiency, they can be utilized in compact hardware implementations of constrained applications, e.g., deeply embedded architectures.

Index Terms—Application-specific integrated circuit (ASIC), fast Fourier transform (FFT), number-theoretic transform (NTT).

I. INTRODUCTION

Number-theoretic transform (NTT) [1] is a discrete Fourier transform defined over a finite ring or field. Being an elegant polynomial multiplication technique, NTT is essential to postquantum cryptosystems, e.g., lattice-based cryptosystems. Such cryptosystems rely on well-studied hard problems, the merit of which is that quantum algorithms to solve these problems efficiently are yet unknown. One of the most common average-case lattice problems is learning with errors problem [2], which assures the hardness of solving other lattice problems in the worst case [3]. However, this very appealing technique gives an impractical key size of quadratic, i.e., $O(n^2)$ complexity, for security parameter $n$ [4]. To reduce the complexity, cyclic [5] and ideal lattices [6] are introduced. Using computation based on fast Fourier transform (FFT), these structures can enable construction of theoretically robust and efficient cryptosystems with quasi-linear, i.e., $O(n \log n)$, key lengths.

Ideal lattices are also employed in fully homomorphic encryption [7] or somewhat homomorphic encryption (SHE) [8], two new primitives with strong potential for securing cloud computing. Polynomial multiplication is the most computationally exhaustive operation of ideal lattices. Applying number theoretic constructions provides a speed advantage, because the polynomial multiplication can be efficiently computed in quasi-linear time $O(n \log n)$ using FFT [9].

Besides postquantum cryptography, NTT can radically improve the currently used schemes by increasing their security parameters. For example, NTT proves to be a valuable tool to signature schemes [10], collision-resistant hash functions [11], as well as identification schemes [12]. As a result, efficient error detection schemes of NTT in polynomial multiplication will boost the security and reliability of postquantum cryptography as well as existing cryptosystems.

Previous studies of NTT-based polynomial multiplication have dealt with reconfigurable hardware [13] and efficient architecture to achieve high speed [14]. Examples of other interesting recent works related to the respective implementations include [15] and [16]. However, no work is yet proposed in the open literature focusing on error detection of NTT polynomial multiplier.

Error detection in cryptography has been the center of attention in previous work [17]–[25]. In this paper, we propose error-detection schemes of NTT polynomial multiplier. The main contributions of this paper are summarized as follows.

1) We introduce a number of categories for error detection in NTT of the ring $\mathbb{R} = (\mathbb{Z}/p\mathbb{Z}[x]/x^n + 1)$. Our proposed schemes are not confined to certain cryptographic constructions.

2) The first category of the proposed error-detection schemes involves recomputing with negated operands. Moreover, we present recomputing with scaled operands. The last category constitutes recomputing with swapped operands. Our target is low hardware overhead, which is favorable to compact and deeply embedded architectures.

3) We implement the proposed error-detection architectures on application-specific integrated circuit (ASIC) for a 65-nm library to assess the implementation and performance metrics. The rest of the paper is organized as follows. Section II reviews the relevant details on efficient computation of NTT. Section III presents our motivation for efficient fault detection as well as our proposed error-detection schemes. Hardware implementations on ASIC along with their overheads are given in Section IV. Finally, Section V concludes this paper.

II. PRELIMINARIES

In this paper, we have considered ideal lattices, defined by $\mathbb{R} = (\mathbb{Z}/p\mathbb{Z}[x]/x^n + 1)$. Here, $f(x)$ is an irreducible polynomial of degree $n$, which can be represented as $f(x) = f_0 + f_1x + f_2x^2 + \ldots + f_{n-1}x^{n-1}$. Also, $n$ is a power of 2, and $p$ is a prime number where $p \equiv 1 \mod 2n$. Multiplication of two polynomials $a(x), b(x) \in \mathbb{Z}_p$, can be represented as: $a(x)b(x) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j x^{i+j} \mod f(x)$, taking quadratic complexity of $O(n^2)$ utilizing school book algorithm.

On the contrary, NTT is a discrete Fourier transform, defined in a finite field, $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}[x]$ [1]. For a given primitive $n$th root of unity in $\mathbb{Z}_p$, $A(x)$ and $B(x)$ are the polynomials under $\mathbb{Z}_p$, where both are generic NTT$_a(a)$ and NTT$_b(b)$, respectively: $A_i = \text{NTT}_a(a(x)i) = \sum_{j=0}^{n-1} a_j a^i j \mod p$, $i = 0, 1, \ldots, n - 1$.
The NTT exists if and only if the block length $n$ divides $q - 1$ for every prime factor $q$ of $p$, where $p$ is a prime and $n$ is a power of 2. Computing Inverse NTT (INTT) is similar to computing NTT, while replacing $\omega$ with $\omega^{-1}$ and introducing $n^{-1}$, i.e., $a_i = \text{INTT}_p^n(a(x)) = n^{-1}\sum_{j=0}^{n-1} a_j\omega^{-ij} \mod p$, $i = 0, 1, \ldots, n - 1$.

As $p$ is a prime, the inverse of $n$, $n^{-1}$ can be computed in modulo $p$, where $nn^{-1} \equiv 1 \mod p$. Applying NTT and INTT to compute polynomial multiplication reduces the time complexity from $O(n^2)$ to $O(n\log n)$.

### III. Proposed Error-Detection Scheme

For high-performance lattice-based cryptography, a flexible NTT-based polynomial multiplier is required. In this section, we present our schemes to provide error-detection hardware architectures with low complexity. The proposed approaches constitute three categories, i.e., recomputing with encoded operands through negated, scaled, and swapped operands.

#### A. Efficient NTT Implementation

In Algorithm 1 [26], the iterative FFT implementation computes the NTT of a given polynomial $a(x) \in \mathbb{Z}_p$. The Bit-Reverse($a$) operation (line 1) reorders the input vector $a$, in which, the new position of the elements in position $k$ can be found by reversing the binary representation of $k$. This algorithm utilizes the “butterfly operation” [21] (lines 8 and 9 of Algorithm 1), which is the multiplication of the polynomial $A[a]$, $d \leftarrow A[b]$, $A[a] \leftarrow c + \omega^N \mod n \cdot d \mod p$, $A[b] \leftarrow c - \omega^N \mod n \cdot d \mod p$.

#### B. Recomputing With Negated Operands

In proposing the error-detection approaches, we make sure that augmenting the original constructions with the proposed schemes leads to low-complexity architectures. As a result, we have applied a number of recomputing with negated operands schemes.

#### C. Recomputing With Scaled Operands

A second variant of the proposed error-detection schemes involves scaling the operands, e.g., doubling, quadrupling, or multiplying with a factor. Let us present an example to explain the scheme. A first example, i.e., recomputing with doubled and quadrupled operands (REEdoQ), involves doubling $\omega$ and $d$, and deriving the quadruple of $c$. The encoded operands would be $A^\prime = 4c + (2o \ast 2d)$ and $B^\prime = 4c - (2o \ast 2d)$. The decoding is performed by dividing the outputs by 4. In binary, dividing by 4 is right shift two places, making decoding a relatively-inexpensive operation. A second example would be, instead of doubling all the operands as REEdoQ,
with swapped operands (RESwO). The recomputed operands are
mod \( k \). Recomputing With Swapped Operands
A both namely, recomputing with scaled dual operand (REScdO), we scale
delay. one right shift operation, resulting in low hardware overhead and time
(REdO). The encoding and decoding of REdO is much similar to
\( \omega \). If we swap \( \omega \) and \( d \), while negating \( c \), we can perform recomputing
with doubled operands (REdO). The encoding and decoding of REdO is much similar to
REdO, requiring only one doubler and one divider, i.e., one left and
doubled only \( \omega \) and \( c \), i.e., recomputing with doubled operands
(doublets), we scale both \( \omega \) and \( c \) by the factor \( k \). This is shown in Fig. 2. The encoding
operations would give \( A' = kc + (k \omega) \ast d = k (c + \omega d) \) and \( B' = kc - (k \omega) \ast d = k (c - \omega d) \). Decoding is performed by dividing both
operands with \( k \) (Fig. 2). As \( p \) is a prime number, \( \gcd(k, p) = 1 \)
mod \( p \), for all values of \( k \).

D. Recomputing With Swapped Operands

If we swap \( \omega \) and \( d \), while negating \( c \), we can perform recomputing
with swapped operands (RESwO). The recomputed operands are
\( A' = -c + \omega d \) and \( B' = -c - \omega d \). As shown in Fig. 3, there is
no necessity for decoding, and RESwO just requires comparison
with alternate prerecomputed values. The only negation unit in the
scheme makes it inexpensive and efficient. We also present a modified
variant of RESwO, i.e., RESwO-m in Fig. 3, in which we lower
the overhead by swapping just \( \omega \) and \( d \), having \( c \) intact. This
would result in even lower overhead as decoding would be free in
hardware.

IV. ASIC Assessments and Comparisons

The proposed error-detection schemes are able to detect transient
and permanent faults (intelligent attackers for intentional/malicious
faults as well as natural defects). In this section, we present the results
of our ASIC assessments using Synopsys Design Compiler and Very
High Speed Integrated Circuit Hardware Description Language with
TSMC 65 nm for three pairs of \((n, p)\) and two of our architectures
to assess the overhead in Table I. We have used Fermat primes in the
form of \( 1 + 2^i \) for \( i = 8, 16, 32 \) which result in having \( \omega = 2 \). Using
65-nm ASIC synthesis, and for three cases \((n, p) = (64, 257), (n, p) = (256, 65537)\),
and \((n, p) = (512, 4294967297)\), we also present the overhead of the presented constructions for the case
studies of the proposed RESwO and RESwO-modified in this paper.
The benchmarking is performed for the error-detection architectures
(for two proposed schemes) and also for the original constructions,
and overheads are shown in parentheses in Table I.
As shown in Table I, the area (in terms of \( \mu m^2 \)), delay (which is an
indication of maximum working frequency), and power consumption
at the frequency of 50 MHz are tabulated. The proposed schemes
achieve acceptable overhead with very high error coverage. One
would use RESwO if both permanent and transient faults in the entire
architectures are to be detected. RESwO-modified has slightly less
overhead and can detect transient faults in the structures.
We have performed simulations for (a) single, (b) two-bit, and
(c) multiple-bit stuck-at-faults. For each experiment, more than
650,000 cases have been considered. From the results, we achieved that
our schemes can detect these three cases with 100% error coverage.
Further analysis shows that if the comparison units (i.e., voters) are
compromised, the error-detection scheme will degrade. Hardening the
comparators, using triple modular redundancy and other fault tolerant
techniques, can solve this faulty comparator status situation.
We would like to finalize this section by noting that the proposed
architectures are oblivious of the standard-cell library and hardware
platform. Therefore, we expect similar results on field-programmable
gate array and ASIC libraries. We also note that the throughput
and frequency overhead can be alleviated through pipelining at the
expense of added hardware overhead.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Area (( \mu m^2 ))</th>
<th>Delay (ns)</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original ((n, p))</td>
<td>2.942</td>
<td>12.24</td>
<td>0.047</td>
</tr>
<tr>
<td>RESwO ((n, p))</td>
<td>3.674</td>
<td>13.37</td>
<td>0.054</td>
</tr>
<tr>
<td>RESwO-(m) ((n, p))</td>
<td>3.544</td>
<td>13.19</td>
<td>0.052</td>
</tr>
<tr>
<td>Original ((n, p))</td>
<td>8.995</td>
<td>13.80</td>
<td>0.093</td>
</tr>
<tr>
<td>RESwO ((n, p))</td>
<td>11.170</td>
<td>14.41</td>
<td>0.111</td>
</tr>
<tr>
<td>RESwO-(m) ((n, p))</td>
<td>11.001</td>
<td>14.23</td>
<td>0.108</td>
</tr>
<tr>
<td>Original ((n, p))</td>
<td>30.829</td>
<td>14.76</td>
<td>0.207</td>
</tr>
<tr>
<td>RESwO ((n, p))</td>
<td>37.476</td>
<td>15.90</td>
<td>0.231</td>
</tr>
<tr>
<td>RESwO-(m) ((n, p))</td>
<td>35.972</td>
<td>15.45</td>
<td>0.228</td>
</tr>
</tbody>
</table>
In this paper, we have presented a number of categories for error-detection schemes of NTT in the ring \( \mathbb{Z}/p\mathbb{Z}[x]/x^n + 1 \), which are also platform-oblivious. The proposed schemes constitute error-detection architectures on hardware based on recomputation with encoded operands. Our target has been low hardware overhead, which is favorable to compact and deeply embedded architectures. We have implemented the proposed error-detection techniques on ASIC for a 65-nm library to assess the implementation and performance metrics. With high error coverage, the presented approaches achieve an 139–158.


