Efficiently Masking Polynomial Inversion at Arbitrary Order

Markus Krausz*[1], Georg Land*[1,2], Jan Richter-Brockmann*[1], and Tim Güneysu[1,2]

[1] Ruhr University Bochum, Horst Görtz Institute for IT Security, Germany
[2] DFKI GmbH, Cyber-Physical Systems, Bremen, Germany
`firstname.lastname@rub.de`

Keywords: PQC, Masking, Polynomial Inversion, Higher-order Masking

Abstract. Physical side-channel analysis poses a huge threat to post-quantum cryptographic schemes implemented on embedded devices. Still, secure implementations are missing for many schemes. In this paper, we present an efficient solution for masked polynomial inversion, a main component of the key generation of multiple post-quantum Key Encapsulation Mechanisms (KEMs). For this, we introduce a polynomial-multiplicative masking scheme with efficient arbitrary order conversions from and to additive masking. Furthermore, we show how to integrate polynomial inversion and multiplication into the masking schemes to reduce costs considerably. We demonstrate the performance of our algorithms for two different post-quantum cryptographic schemes on the Cortex-M4. For NTRU, we measure an overhead of 35 % for the first-order masked inversion compared to the unmasked inversion while for BIKE the overhead is as little as 11 %. Lastly, we verify the security of our algorithms for the first masking order by measuring and performing a TVLA based side-channel analysis.

1 Introduction

Our digital infrastructure relies and trusts Public-Key Cryptography (PKC) to establish secure communication channels. However, due to Shor's algorithm presented in 1999 [36], currently used schemes like RSA [33] and ECC [29] can be broken by quantum computers in polynomial time. Therefore, in 2017, the National Institute of Standards and Technology (NIST) announced a *Post-Quantum Cryptography Standardization Project* to find and standardize new cryptographic schemes that provide security against attacks mounted on classical and quantum computers. After three rounds, the NIST identified seven finalists and eight alternate candidates which are considered for standardization. Besides security, important metrics like costs, performance, and implementation characteristics on various platforms are considered in the selection process [2]. Driven by these

* These authors contributed equally to this work

criteria, the research community has proposed a plethora of highly efficient implementations for software and hardware. However, implementations of Post-Quantum Cryptography (PQC) schemes on embedded devices are faced with the same problems as traditional cryptographic algorithms, which includes physical attacks like Side-Channel Analysis (SCA) and Fault-Injection Analysis (FIA).

So far, most of the side-channel research with respect to the finalists in NIST's PQC standardization process focuses on schemes based on the Learning with Error (LWE) problem. Bos et *al.* presented the first higher-order masked implementation for the Cortex-M0+ and the Cortex-M4 for Kyber [8]. Just recently, Heinz et *al.* published a report on an optimized first-order protected Kyber implementation for the Cortex-M4 including practical measurements [19]. In 2021, Beirendonck et *al.* presented a first-order protected implementation of Saber for the Cortex-M4 [4]. An optimized implementation that also provides protection against higher-order attacks was afterwards proposed in [26].

Besides these studies that directly target the protection of specific algorithms, others [14,18] proposed optimizations and implementations which can be applied to both schemes. Coron et *al.* [14] concentrated their work on the improvements of higher-order masked comparisons by considering different approaches and techniques. As a case study, they applied their optimizations to Kyber and Saber. The work of Fritzmann et *al.* [18] explored different masked accelerators used as instruction set extensions for a RISC-V processor. They demonstrated their improvements on a hardware software co-design for Kyber and Saber. Eventually, D'Anvers et *al.* improved the work of Coron et *al.* [14] and presented an optimized higher-order masked comparison [15].

Summarizing, we can see that the side-channel security countermeasures for the LWE problem based schemes Kyber and Saber have already received some attention. However, masking NTRU-like [20, 21] and code-based [3, 28] systems is still an open research question and has so far only been sparsely investigated. In contrast, several side-channel attacks on these schemes were demonstrated. At CHES 2019, Sim et *al.* present a generic side-channel attack using conditional moves in implementations of PQC schemes based on Quasi-Cyclic Moderate-Density Parity-Check (QC-MDPC) codes [37]. Recently, a single-trace side-channel attack on the polynomial sampling of NTRU, NTRU Prime, and Dilithium has been proposed in [25]. In the work of Mujdei et *al.* [30] the authors present a powerful correlation power analysis on polynomial multiplications effecting all lattice-based PQC schemes.

An important operation in almost all NTRU-like and code-based systems is the polynomial inversion. It is required in the key generation of the finalists NTRU-HPS and NTRU-HRSS [10] as well as in the two alternate candidates Streamline NTRU Prime [5] and BIKE [3].

Contribution. To this end, we present the first efficient methodology for masking polynomial inversion by introducing polynomial-multiplicative masking (Section 3). As a foundation for our approach, we develop secure arbitrary-order conversions from polynomial-additive to polynomial-multiplicative masking (Section 3.1) and vice versa (Section 3.2). We show how to integrate a masked polynomial inversion into this conversion to reduce the number of unmasked inversions to one, independent of the masking order (Section 3.3). Additionally, we develop an algorithm to integrate a masked polynomial multiplication into the conversion to save costly unmasked multiplications (Section 3.4). Finally, we implement our algorithms for two use cases to demonstrate the performance benefits and we back our security claims for the first masking order by performing practical measurements on a Cortex-M4 microcontroller (Section 4).

2 Preliminaries

In this section we introduce important preliminaries that are necessary to adequately describe our approaches of masked arithmetic operations. Besides stating notations used throughout this work, we briefly recap masking. Eventually, we describe practical applications of masked polynomial inversions in the field of PQC.

2.1 Notation

Throughout this work, we denote polynomials by x. The i-th share of a shared polynomial x is denoted by x_i. A uniform random sampling of a polynomial r is denoted by $r \overset{\$}{\leftarrow} \mathcal{R}$ where $\mathcal{R}$ is the set of all valid polynomials. The set $\mathcal{R}^*$ denotes all uniform sampled polynomials from $\mathcal{R}$ that are invertible.

2.2 Masking

Masking is a common countermeasure to prevent SCA on embedded devices and is studied in the scientific community for more than twenty years [9]. The foundation of masking is *secret sharing* which splits a sensitive value x into multiple shares x_i with $0 \leq i \leq d$. For a correct sharing holds

$$x = x_0 \circ x_1 \circ \cdots \circ x_d \tag{1}$$

where $\circ$ defines the group operator of the applied masking scheme and d defines the security order based on the d-probing model proposed in [22]. As a consequence, a function f processing x needs to be transformed as well such that $f = f_0 \circ f_1 \circ \cdots \circ f_d$. When applying $\oplus$ as the group operator in Equation 1, the secret sharing scheme is called *boolean masking.* The encoding is called *arithmetic masking* when $\circ$ is replaced by an addition or multiplication which we further categorize as *additive masking* or *multiplicative masking*, respectively.

2.3 Polynomial Inversion Applications

Polynomial inversion is a regular used operation in several PQC schemes [3, 20, 21, 28]. Since it is such a critical operation, several works concentrated on efficient implementations of the polynomial inversion for software and hardware [11, 17, 31, 32]. However, most approaches are based on Fermat's Little Theorem performed by the Itoh-Tsujii Algorithm (ITA) algorithm [23] or on the extGCD proposed by Bernstein and Yang [7]. In the following, we will briefly introduce the finalist NTRU, and the two alternate candidates streamlined NTRU Prime and Bit Flipping Key Encapsulation (BIKE) as examples of PQC schemes requiring polynomial inversions.

NTRU. The finalist NTRU is based on the original work by Hoffstein et *al.* [20] and on the work by Hülsing et *al.* [21]. NTRU is defined by three coprime positive integers (n, p, q), the sample spaces $\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r, \mathcal{L}_m$, and an injection $\mathsf{Lift} : \mathcal{L}_m \to \mathbb{Z}[\mathbf{X}]$. Furthermore, the authors of the NTRU submission recommend two families of parameter sets called NTRU-HPS and NTRU-HRSS [10]. NTRU-HPS uses a *fixed-weight* sampling space and allows several choices of q for each n which are based on [20] while NTRU-HRSS uses an *arbitrary weight* sampling space and fixed q as a function of n as suggested in [21].

The key generation requires to perform two polynomial inversions to generate the public and private key as shown in Algorithm 1. Note, for NTRU-HPS as well as for NTRU-HRSS the parameter p is always fixed to three. However, the two parameters (n, q) are different for the three security levels $\lambda \in \{1, 3, 5\}$ and are defined as $(509, 2\,048)$, $(677, 2\,048)$, and $(821, 4\,096)$, respectively.

Streamlined NTRU Prime. Streamlined NTRU Prime [5] is an alternate candidate in the NIST standardization process. NTRU Prime is also based on the original proposal by Hoffstein et *al.* [20] and defined by a prime number p, a prime number q, and a positive integer w [6]. One of the main differences to the classic NTRU cryptosystem is that NTRU Prime works over prime fields which avoids various attack vectors as claimed by the authors [5]. The key generation in NTRU Prime (see Algorithm 2) also contains two polynomial inversions. The first inversion inverts the randomly sampled polynomial g drawn from R while the second inversion inverts $3 \cdot f$ where f is a polynomial with coefficients $f_i \in \{-1, 0, 1\}$ with exactly w non-zero coefficients. Note, the first sampled polynomial g is not always invertible in R_3 while the second polynomial f is always invertible in R_q since it is a field.

For the three security levels $\lambda \in \{1, 3, 5\}$ the NTRU Prime parameters (p, q, w) are defined as $(653, 4\,621, 288)$, $(953, 6\,343, 396)$, and $(1\,277, 7\,879, 492)$, respectively.

Algorithm 1 KeyGen NTRU.

```
Require: NTRU parameters $n, p, q$.
Ensure: Priv. key $(f, f_p, f_q)$, pub. key $h$.
 1: Generate $f \overset{\$}{\leftarrow} \mathcal{L}_f$
 2: Generate $g \overset{\$}{\leftarrow} \mathcal{L}_g$
 3: Compute $f_p \leftarrow 1/f$ in $S_3$
 4: Compute $f_q \leftarrow 1/f$ in $S_q$
 5: Compute $g \leftarrow 3 \cdot g \cdot f_g$ in $R_q$
 6: Compute $h_q \leftarrow 1/h$ in $S_q$
 7: Return $(f, f_p, f_q)$ and $h$.
```

Algorithm 2 KeyGen sNTRUp.

```
Require: sNTRUp parameter $q$.
Ensure: Priv. key $(f, g_{inv})$, pub. key $h$.
  repeat
      Generate $g \overset{\$}{\leftarrow} R$, $g$ small
  until $g$ is invertible in $R_3$
  Generate $f \overset{\$}{\leftarrow} Short$
  Compute $g_{inv} \leftarrow 1/g$ in $R_3$
  Compute $h \leftarrow g/(3 \cdot f)$ in $R_q$
  Return $(f, g_{inv})$ and $h$.
```

BIKE. As well as Streamlined NTRU Prime, BIKE has been selected as an alternate candidate. In contrast to NTRU, BIKE is a code-based scheme relying on QC-MDPC codes [3]. The scheme originally consists of three different algorithms BIKE-1, BIKE-2, and BIKE-3 which, however, were reduced to just one single Key Encapsulation Mechanism (KEM) called BIKE. In BIKE, all polynomials are from the cyclic polynomial ring $\mathcal{R} := \mathbb{F}_2[X]/(X^r - 1)$ where r defines the size of the polynomials. The public key h is generated by sampling two private sparse polynomials (h_0, h_1) with $|h_0| = |h_1| = w/2$, inverting h_0, and multiplying the results with h_1. The entire key generation if formally described in Algorithm 3. For the three security levels $\lambda \in \{1, 3, 5\}$, the two parameters (r, w) are defined as $(12\,323, 141)$, $(24\,659, 206)$, and $(40\,973, 274)$, respectively. Since BIKE is suggested to be used with ephemeral keys, an efficient masked implementation of the polynomial inversion for side-channel protected designs is necessary.

In summary, it can be seen in Algorithm 1, Algorithm 2, and Algorithm 3 that the polynomial inversion is a major operation in the key generation of all three algorithms. Our measurements in Section 4.1 confirm that the polynomial inversion dominates the costs in terms of cycle counts. Hence, to construct protected designs against SCA, it is essential to find efficient algorithms for masked implementations. However, not only the inversion itself should be implemented efficiently but also preceding and subsequent operations must be masked without any expensive conversions between different masking techniques. Before we present our approach of an efficient higher-order masked polynomial inversion, we briefly discuss different cases of invertibility of random polynomials.

Invertibility of Random Polynomials Among these three schemes, three different cases of invertibility occur. Since the target polynomials are sampled randomly but based on certain rules, we identify the following cases.

Algorithm 3 Key Generation of BIKE.

```
Require: BIKE parameters $n, w, \ell$.
Ensure: Private key $(h_0, h_1, \sigma)$ and public key $h$.
  Generate $(h_0, h_1) \xleftarrow{\$} \mathcal{R}^2$ both of odd weight $|h_0| = |h_1| = w/2$.
  Generate $\sigma \xleftarrow{\$} \{0,1\}^\ell$ uniformly at random.
  Compute $h \leftarrow h_1 h_0^{-1}$.
  Return $(h_0, h_1, \sigma)$ and $h$.
```

1. All sampled polynomials (except the polynomial representing 0) are invertible. This case is trivial and no further exceptions need to be covered which is the case for NTRU.
2. Not all polynomials from the used ring are invertible but following some certain rules always allows to sample an invertible polynomial. For example, this is the case for BIKE where the polynomials requires to have an odd Hamming weight. Hence, applying the defined sampling procedure always generates an invertible polynomials such that the inversion cannot fail.
3. Not all polynomials from the underlying ring are invertible and they are not easily distinguishable. For example, this is the case for Streamlined NTRU Prime where the sampling procedure just sample uniformly random polynomials without applying dedicated rules. In case the sampled polynomial is not invertible, the inversion fails in the last step and a new polynomial needs to be sampled.

3 Masking Polynomial Inversion

Masking boolean operations in PQC schemes can efficiently be implemented with a boolean sharing, while arithmetic operations such as the addition and subtraction of polynomials or the multiplication with public values are implemented with additive sharing as the masked implementation for Kyber [8] demonstrates. An alternative sharing, that had already been proposed for AES in the year 2001 [1], is multiplicative sharing. The problem with multiplicative sharing that hinders its application, is that if one share is zero, the attacker already knows that the masked value is zero.

For polynomial inversion, that is used in multiple PQC schemes as shown in Section 2.3, we need a masking approach for which inversion is a linear operation. Given uniformly random polynomials $m_i \in \mathcal{R}$ such that $m = \prod_{i=0}^{d} m_i$, a valid polynomial multiplicative sharing can be realized by

$$m^{-1} = \prod_{i=0}^{d} m_i^{-1}, \tag{2}$$

i.e., the inversion is applied to each share independently. As the zero polynomial is not invertible, it will not be given as an input to a masked inversion. With

$d+1$ unmasked polynomial inversions, that is already an expensive operation on its own, this approach is very costly and asks for alternative solutions.

Obviously, multiplication of *two secret* polynomials is very efficient in the multiplicative domain as it requires only $d+1$ unmasked multiplications compared to the additive domain where the number of unmasked multiplications is quadratic to the masking order in current solutions [35]. The cost to convert polynomials from and to the multiplicative domain determines, however, whether this approach is viable (cf. Section 3.4).

In the following, we present algorithms that efficiently transform additive shares of polynomials in a ring to multiplicative shares and vice versa. With the motivation to perform a more efficient polynomial inversion than shown in Equation 2, we demonstrate how to integrate the inversion into the transformation, and how to perform a multiplication and back transformation in one joint operation.

3.1 Conversion from Additive to Multiplicative Sharing

Let a be a polynomial and a_i shares with $a = \sum_{i=0}^{d} a_i$, where all a_i are uniform random in the respective polynomial ring. To transform this sharing to a polynomial-multiplicative sharing in the same ring, we adapt the well-known technique of first appending a share in the new masking domain, enlarging the sharing in two domains (additive and multiplicative), and then to combine two old shares to remove one share.

We now introduce our algorithm by presenting an example for first-order masking. Given a polynomial a split into two additive shares a_0 and a_1, we start by sampling one invertible polynomial r and multiply each additive share with this polynomial, yielding ra_0 and ra_1. We set the inverted polynomial r^{-1} as a new multiplicative share, expanding the number of shares from two to three. To reduce our number of shares, we add corresponding two additive shares: $ra_0 + ra_1 = r(a_0 + a_1)$. By treating the sum as a multiplicative share, we are left with two correct multiplicative shares for a, since $r^{-1}r(a_0 + a_1) = a$.

The full algorithm for arbitrary orders can be summarized with the following steps:

1. Sample a uniform random and invertible polynomial r, observing that $a = r^{-1}ra$.
2. Compute $a'_i = ra_i$, we now have $d+2$ shares, $d+1$ additive shares and one multiplicative share.
3. To return to $d+1$ shares, we combine two additive shares.
4. Repeat from start until there is only one additive share left, which now can be viewed as a multiplicative share.

The algorithm is shown in detail in Algorithm 4. Note that for this conversion, d polynomial inversions and $(d+1)(d+2)/2 - 1$ polynomial multiplications, as well as $d-1$ polynomial additions are needed.

Algorithm 4 Additive to Polynomial Multiplicative Masking Conversion (A2M)

```
Require: $a = \sum_{i=0}^{d} a_i$
Ensure: $m = \prod_{i=0}^{d} m_i = a$
  function A2M($a_0, \dots, a_d$)
    for $i := d$ downto 1 do
      $r \xleftarrow{\$} \mathcal{R}^*$                ▷ sample from the set of invertible polynomials
      for $j := 0$ to $i$ do
        $a_j := r a_j$
      end for
      $m_i := r^{-1}$        ▷ now we have $d + 2$ shares with $a = \left(\sum_{j=0}^{i} a_j\right) \prod_{j=i}^{d} m_j$
      $a_{i-1} := a_{i-1} + a_i$                        ▷ combining two additive shares
    end for
    $m_0 := a_0$
  end function
```

3.2 Conversion from Multiplicative to Additive Sharing

For subsequent operations in the additive domain, a transformation from the multiplicative to the additive domain is necessary. Given a masked polynomial m split into two multiplicative shares m_0 and m_1 for our M2A conversion, we start by sampling one random polynomial r. The first step is to compute $m_0 + r$ before we multiply it with m_1 to get $(m_0 + r)m_1 = m_0 m_1 + r m_1$. Together with the product $-r m_1$ we have two additive shares that yield $m_0 m_1 + r m_1 - r m_1 = m_0 m_1 = m$.

This method can be generalized to arbitrary masking orders by reapplying the core idea of adding a random polynomial before the multiplication with one of the multiplicative shares m_i. Our strategy is to compute $m = \prod_{i=0}^{d} m_i$ step by step in the first share, while protecting this sum with d random summands. Thus, iterating from $i = 1$ to d, we sample a uniform random additive sharing of $i + 1$ polynomials such that $\sum_{j=0}^{i} r_{ij} = 0$. We add these random polynomials to the first $i + 1$ shares before we multiply the shares with m_i. After d iterations, we get $a_0 = m + \sum_{i=1}^{d} (r_{i0} \prod_{j=i}^{d} m_j)$ as the first additive share for m together with d additive shares $a_k = \sum_{i=k}^{d} (r_{ik} \prod_{j=i}^{d} m_j)$ that cancel out the summands in a_0 except m.

The algorithm can efficiently be implemented in situ as shown in Algorithm 5 and utilizes $d(d + 1)/2 + d$ polynomial multiplications, $d(d + 1) + d$ additions, $d(d + 1)/2$ fresh random polynomials and no costly inversion.

3.3 Reducing the Number of Inversions

The main application of the polynomial-multiplicative masking is polynomial inversion. Naively, we would perform a polynomial inversion on each polynomial-multiplicative share individually to obtain a sharing of the inverted polynomial (cf. Equation 2). Together with the d inversions required for the A2M conversion,

Algorithm 5 Polynomial-Multiplicative to Additive Masking Conversion (M2A)

```
Require: $m = \prod_{i=0}^{d} m_i$
Ensure: $a = \sum_{i=0}^{d} a_i = m$
  function M2A($m_0, \ldots, m_d$)
      for $i := 1$ to $d$ do
          $r_i := 0$
          for $j := 0$ to $i - 1$ do
              $r \xleftarrow{\$} \mathcal{R}$
              $r_i := r_i + r$
              $m_j := m_j + r$                                    ▷ refreshing
              $m_j := m_j m_i$                ▷ combining two multiplicative shares
          end for
          $m_i := -r_i m_i$
      end for
      for $i := 0$ to $d$ do
          $a_i := m_i$
      end for
  end function
```

this would lead to $2d + 1$ unmasked inversions for one masked inversion, given a polynomial shared in the additive domain.

However, we can adapt Algorithm 4 such that only one polynomial inversion is necessary, independent of the masking degree. This is shown in Algorithm 6. The idea is to not set the new multiplicative shares to the inverse, which we would invert again later, but to the original sample. Instead we only invert m_0 at the end to get an A2M conversion with implicit inversion. With this method we can drastically reduce the number of polynomial inversions that are the most expensive operations compared to polynomial multiplications and additions as we show in Section 4. We thus save two inversions for first order, four inversions for second and already six inversions for third order masking, compared to the naive approach.

3.4 Reducing the Number of Multiplications

Although a masked polynomial multiplication is cheaper in the multiplicative domain ($d + 1$ unmasked multiplications) compared to the additive domain where it is quadratic [35], the additional costs of the A2M and M2A conversions render this approach obsolete for polynomials that are not given in the multiplicative domain anyway. In particular the A2M conversion without inversion is too expensive with its d unmasked inversions.

We can, however, save unmasked multiplications when one factor is already in the multiplicative domain due to a prior inversion. Given a polynomial $a = \sum_{i=0}^{d} a_i$ in the additive domain and a polynomial $b = \prod_{i=0}^{d} b_i$ in the multiplicative domain, we observe that the masked product $c = \sum_{i=0}^{d} c_i = ab$ can be computed with $c = ab = \sum_{i=0}^{d} a_i \prod_{j=0}^{d} b_j = \sum_{i=0}^{d} (a_i \prod_{j=0}^{d} b_j)$, where

Algorithm 6 Additive to Polynomial Multiplicative Masking Conversion with Implicit Polynomial Inversion ($\mathsf{A2M}_{\mathrm{INV}}$)

```
Require: $a = \sum_{i=0}^{d} a_i$
Ensure: $m = \prod_{i=0}^{d} m_i = a^{-1}$
  function $\mathrm{A2M}_{\mathrm{INV}}(a_0, \ldots, a_d)$
    for $i := d$ downto 1 do
      $r \xleftarrow{\$} \mathcal{R}^*$                    ▷ sample from the set of invertible polynomials
      for $j := 0$ to $i$ do
        $a_j := r a_j$
      end for
      $m_i := r$                          ▷ note that we do not set this to the inverse
      $a_{i-1} := a_{i-1} + a_i$
    end for
    $m_0 := a_0^{-1}$                          ▷ the only inverse we need to compute
  end function
```

$c_i = a_i \prod_{j=0}^{d} b_j$ represents an additive share of the product c. The straightforward computation would leak the polynomial b, but by adding fresh random polynomials between the unmasked multiplications similar as in our $\mathsf{M2A}$ conversion, we can get a secure conversion from multiplicative domain to additive domain including a multiplication with an additive shared polynomial as shown in Algorithm 7.

The costs for this masked conversion with implicit multiplication are $(d+1)^2$ unmasked multiplications, $(d+1)2d$ additions and $(d+1)d$ fresh random polynomials. Compared to the naive approach of first converting a from the multiplicative to the additive domain and then performing the multiplication, we save about the amount of unmasked multiplications and additions required for the $\mathsf{M2A}$ conversion.

For the case where we want to securely invert a polynomial and multiply the result with another polynomial, which is often the case as we see in Section 2.3, we apply our $\mathsf{A2M}_{\mathrm{INV}}$ first, where the costs are dominated by the single unmasked inversion, resulting in an inverted polynomial in the multiplicative domain. As a second step, we apply our $\mathsf{M2A}_{\mathrm{MUL}}$, to transform the inverted polynomial back into the additive domain while simultaneously multiplying it with another additive shared polynomial, at the cost of a multiplication in the additive domain, so the back transformation is basically free. In Section 4 we present performance results by exemplary applying our approaches to NTRU and BIKE.

4 Implementation and Evaluation

To evaluate the performance and security of our algorithms, we implemented them for NTRU and BIKE on the STM32F4 discovery board, which is equipped with a 32-bit Cortex-M4 microcontroller, 192-KB SRAM and 1-MB flash memory and can be clocked up to 168 MHz.

Algorithm 7 Polynomial-Multiplicative to Additive Masking Conversion with Implicit Polynomial Multiplication ($\mathsf{M2A}_{\mathrm{MUL}}$)

```
Require: $a = \sum_{i=0}^{d} a_i$, $b = \prod_{i=0}^{d} b_i$
Ensure: $c = \sum_{i=0}^{d} c_i = ab$
  function $\mathrm{M2A}_{\mathrm{MUL}}(a_0, \ldots, a_d, b_0, \ldots, b_d)$
    for $j := 0$ to $d$ do
      $c_j := a_j b_0$                              ▷ implicit multiplication
    end for
    for $i := 1$ to $d$ do
      $r_d := 0$
      for $j := 0$ to $d - 1$ do
        $r \xleftarrow{\$} \mathcal{R}$
        $r_d := r_d + r$
        $c_j := c_j + r$                            ▷ refreshing
        $c_j := c_j b_i$                            ▷ combining two multiplicative shares
      end for
      $c_d := c_d - r_d$
      $c_d := c_d b_i$
    end for
  end function
```

We based our implementation on the respective ring operations of the state-of-the-art Cortex-M4 implementations of the schemes. For BIKE this is the soon to be published work by Chen et *al.* [12], for NTRU this is the work by Chung et *al.* [13] with an improved inversion by Li et *al.* [27].

4.1 Implementation Results

As it is common [24], we measured cycle counts at 24 MHz to not have memory wait states. We compiled our code with the `arm-none-eabi-gcc-10.3.1` compiler with optimization-level `O3`. The stated cycle counts are averages of 100 runs.

We did not implement and measure the plain A2M conversion, because it is not interesting for our use cases with its high costs.

NTRU. We first measured the cycle counts for unmasked ring operations to have a baseline to compare our masked versions with. For NTRU in the parameter set `ntruhps2048677`, polynomials in the ring S_3 have 677 coefficients $\in \{0, 1, 2\}$. Unprotected polynomial inversion costs 1 273 864 clock cycles here, about six times the cycles for an unprotected polynomial multiplication that takes 201 383 cycles. An unprotected addition is done in only 18 340 cycles and is thus insignificant compared to inversions and multiplications.

The costs for the masked $\mathsf{A2M}_{\mathrm{INV}}$ in the first masking order are mainly determined by the unmasked inversion and two unmasked multiplications. The overhead compared to an unmasked inversion is therefore mostly the cost of two multiplications, resulting in about 35 % overhead, which is an excellent result compared to other masked operations. This calculation excludes the cost of an M2A conversion, but as we argued in Section 3.4, this comes for free by using the $\mathsf{M2A}_{\mathrm{MUL}}$. Since the number of unmasked inversions required for one $\mathsf{A2M}_{\mathrm{INV}}$ is only one, independent of the masking order, the cycle counts of the $\mathsf{A2M}_{\mathrm{INV}}$ increase only slowly with the masking order. For the sixth order, which operates on seven shares, the cycle counts are less than six fold the ones of the unmasked as shown in Table 1.

For the $\mathsf{M2A}_{\mathrm{MUL}}$ we measured 885 773 cycles in the first order, less than twice the cost of one M2A that costs 486 165. This proportion stays with increasing masking order while the number of unmasked multiplications and additions grows quadratically for both algorithms.

Table 1: Cycle counts for our proposed masked $\mathsf{A2M}_{\mathrm{INV}}$, $\mathsf{M2A}_{\mathrm{MUL}}$, and M2A conversion for ntruhps2048677 on the Cortex-M4. Unprotected addition requires 18 340 clock cycles, unprotected multiplication requires 201 383 clock cycles and unprotected inversion 1 273 864 clock cycles.

	A2M Inversion			**M2A Mul.**			**M2A Conversion**		
Ord. d	Cycles	MUL	INV	Cycles	MUL	ADD	Cycles	MUL	ADD
1	1 723 778	2	1	885 773	4	4	486 165	2	3
2	2 372 502	5	1	2 090 841	9	12	1 230 767	5	8
3	3 211 410	9	1	3 802 004	16	24	2 238 833	9	15
4	4 260 732	14	1	6 057 128	25	40	3 503 189	14	24
5	5 524 861	20	1	8 848 501	36	60	5 049 140	20	35
6	6 991 050	27	1	12 097 869	49	84	6 859 272	27	48

BIKE. For BIKE in the parameter set `bikel1`, polynomials have 12 323 coefficients $\in \{0, 1\}$. As 32 coefficients are stored in one register and the addition of coefficients equates to a xor operation, the unmasked addition of polynomials is very cheap with 3 534 clock cycles. Due to the higher polynomial degree, however, multiplications and inversions take longer, compared to the operations in NTRU. For one unmasked multiplication, we measured about one million cycles, and for one unmasked inversion 19 182 916 cycles.

With the increased gap between multiplication and inversion, compared to NTRU, the overhead of the $\mathsf{A2M}_{\mathrm{INV}}$ reduces. With 21 317 392 cycles for the first order $\mathsf{A2M}_{\mathrm{INV}}$, the overhead is as little as 11 % compared to an unmasked inversion. Also the cost of $\mathsf{M2A}_{\mathrm{MUL}}$ and M2A become less significant compared a $\mathsf{A2M}_{\mathrm{INV}}$ in the lower masking orders, due to the order of magnitude difference in cycle counts between unmasked inversion and unmasked multiplication. In the first masking order we measure 4 240 017 cycles for one $\mathsf{M2A}_{\mathrm{MUL}}$ and 2 131 405 for one M2A as shown in Table 2. The gap between $\mathsf{A2M}_{\mathrm{INV}}$ and M2A or $\mathsf{M2A}_{\mathrm{MUL}}$

decreases in relative terms with increasing masking order due to the quadratic cost in unmasked multiplications.

Table 2: Cycle counts for our proposed masked $\mathsf{A2M}_{\mathrm{INV}}$, $\mathsf{M2A}_{\mathrm{MUL}}$, and $\mathsf{M2A}$ conversion for bikel1 on the Cortex-M4. Unprotected addition requires 3 534 clock cycles, unprotected multiplication requires 1 052 253 clock cycles and unprotected inversion 19 182 916 clock cycles.

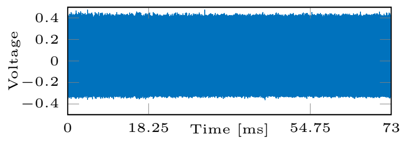
	A2M Inversion			**M2A Mul.**			**M2A Conversion**		
Ord. d	Cycles	MUL	INV	Cycles	MUL	ADD	Cycles	MUL	ADD
1	21 317 392	2	1	4 240 017	4	4	2 131 405	2	3
2	24 487 146	5	1	9 584 999	9	12	5 342 630	5	8
3	28 736 397	9	1	17 068 753	16	24	9 622 491	9	15
4	34 007 250	14	1	26 740 596	25	40	14 994 627	14	24
5	40 275 530	20	1	38 507 790	36	60	21 419 851	20	35
6	47 744 390	27	1	52 493 255	49	84	28 945 019	27	48

4.2 Side-Channel Evaluation

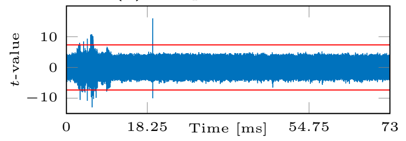
To evaluate the security against power side-channel attacks, we performed measurements on the same STM32F4 discovery board with the Cortex-M4 microcontroller. The power consumption is indirectly measured via a $1\,\Omega$ shunt resistor placed in the supply path of the microcontroller (the board provides dedicated pads for such applications) and the signal is amplified by a ZFL-1000LN+ Low Noise Amplifier (LNA). We use an 8 bit oscilloscope from PicoScope sampling with 625 MS/s to acquire the power traces. During the measurements, the microcontroller operates with a 24 MHz clock, which results in roughly 26 sample points per clock cycle, and is powered by an external power supply to ensure a clean and stable supply voltage.

For the security evaluation, we use a common fixed vs. random univariate Test Vector Leakage Assessment (TVLA) evaluation procedure as detailed described in [34]. Commonly, the measured power traces of the fixed and random inputs are used for a Welsh t-test where the t-value is compared to a ± 4.5 threshold corresponding to a $\alpha = 0.000\,1$ confidence level. In case the threshold is exceeded, the implementation is assumed to leak sensitive information since the power consumption of the fixed and the random inputs can be distinguished. However, in 2017 Ding et *al.* demonstrated that the threshold of ± 4.5 needs to be adapted for measurements with many sample points to avoid false positives in the evaluation [16]. Since we measure operations that require up to 1.7×10^6 clock cycles (which are approximately $26 \cdot 1.7 \times 10^6 = 44.2 \times 10^6$ sample points with our setup), we applied their approach and adapted the corresponding threshold that still results in a confidence level of α.

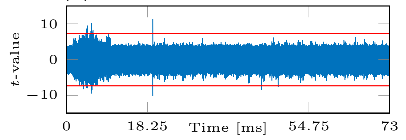
In the following, we present the measurement results for the first-order masked inversion $\mathsf{A2M}_{\mathrm{INV}}$ and the multiplicative to additive conversion $\mathsf{M2A}$.



(a) Sample trace.

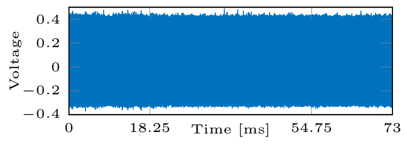


(b) First-order t-test results.

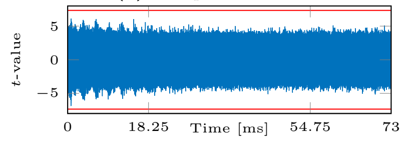


(c) Second-order t-test results.

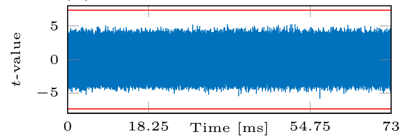
Fig. 1: Measurement results of $\mathsf{A2M}_{\text{INV}}$ with no randomness (2 000 traces).



(a) Sample trace.



(b) First-order t-test results.

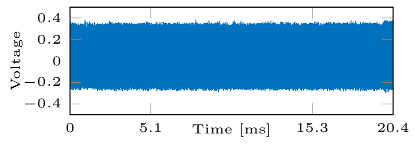


(c) Second-order t-test results.

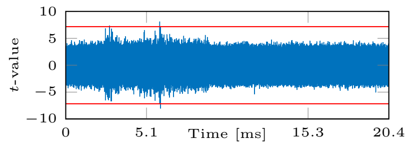
Fig. 2: Measurement results of $\mathsf{A2M}_{\text{INV}}$ with randomness (100 000 traces).

We limit our evaluation to these two algorithms as they exemplary demonstrate the ideas of our proposals. Both, the $\mathsf{A2M}$ conversion and the $\mathsf{M2A}_{\text{MUL}}$, are similar to the other two algorithms such that we only performed the time-consuming measurements for them.

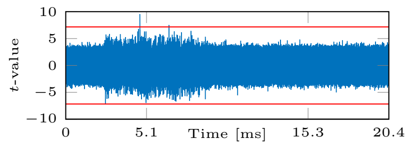
Masked Inversion. Figure 1 shows the measurement results for the masked inversion presented in Algorithm 6 with disabled randomness to demonstrate the correct functionality of our measurement setup. As expected, the t-test reveals first- and second-order univariate leakage. Figure 2 presents the measurement results for the protected inversion with randomness enabled. We acquired 100 000 power traces and could not detect any first-order univariate leakage. Interestingly, the second-order t-test also does not reveal any leakage which is may due to the univariate analysis technique applied in our evaluation. We expect that second-order leakage would be visible once an attacker utilizes multivariate analysis techniques, i.e., combines samples from multiple points in time. Another reason for this phenomena could be the applied masking technique. When we look at a single coefficient of a polynomial with multiplicative sharing, it can not be recreated by $d+1$ respective coefficients of the polynomial shares, but depends on other coefficients too. For the first masking order we combine one random coefficient of one polynomial with all random coefficients of another polynomial which can be seen as some kind of higher order masking. However, this artifact is out of scope of this work and we leave the investigation for future work.



(a) Sample trace.

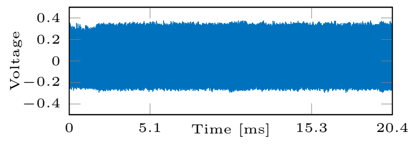


(b) First-order *t*-test results.

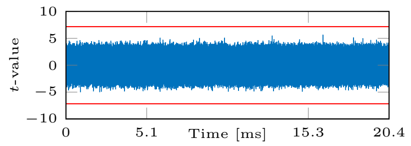


(c) Second-order *t*-test results.

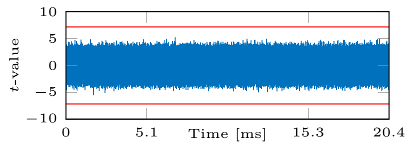
Fig. 3: Measurement results of the M2A conversion with no randomness (2 000 traces).



(a) Sample trace.



(b) First-order *t*-test results.



(c) Second-order *t*-test results.

Fig. 4: Measurement results of the M2A conversion with randomness (100 000 traces).

Multiplicative to Additive Conversion. Besides the masked polynomial inversion, we additionally evaluate the multiplicative to additive conversion M2A from Algorithm 5. Again, we first measured the operation with disabled randomness (masks and fresh randomness are constant) which is visualized in Figure 3. After 2 000 traces, the *t*-test results for the first- and second-order clearly indicate leakage. However, in the next experiment we enable all randomness and perform 100 000 measurements. The *t*-test does not reveal any leakage which is shown in Figure 4. Again, no second-order leakage is visible due to the same argumentation as above.

5 Conclusion

In this work, we demonstrate that polynomial-multiplicative sharing is a viable solution to mask arithmetic operations of multiple PQC schemes. To this end, we propose an efficient higher-order masked polynomial inversion with implicit additive to multiplicative conversion, conversion algorithms used to switch between different sharings, and a novel masked multiplication that accepts an additive shared operand and a multiplicative shared operand. Applying our masked polynomial inversion to NTRU, the first-order masked design requires an overhead of only 35 %, while the overhead for BIKE is only 11 %.

However, there are still masking solutions missing for other operations to have all the pieces necessary for a masked implementation of NTRU or BIKE, which is an interesting target for future work. Another open question is the additional security that polynomial-multiplicative masking provides, when looking at the coefficient level. As already mentioned in Section 4.2, traditional masking schemes split one value into $d+1$ values. But in polynomial multiplication, all coefficients are combined with each other and make one coefficient of the masked polynomial dependent of more than $d+1$ values.

Acknowledgments

This work was supported by the German Research Foundation under Germany's Excellence Strategy – EXC 2092 CASA – 390781972, through the H2020 project PROMETHEUS (grant agreement ID 780701), and by the Federal Ministry of Education and Research of Germany through the QuantumRISC (16KIS1038) and PQC4Med (16KIS1044) projects.

References

1. Mehdi-Laurent Akkar and Christophe Giraud. An Implementation of DES and AES, Secure against Some Attacks. In *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.
2. Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, et al. *Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process*. US Department of Commerce, National Institute of Standards and Technology, 2019. `https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=927303`.
3. Nicolas Aragon, Paulo SLM Barreto, Slim Bettaieb, France Worldline, Loïc Bidoux, Olivier Blazy, Philippe Gaborit, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, et al. BIKE: Bit Flipping Key Encapsulation. 2021. `https://bikesuite.org/files/v4.2/BIKE_Spec.2021.07.26.1.pdf`.
4. Michiel Van Beirendonck, Jan-Pieter D'anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A Side-channel Resistant Implementation of SABER. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 17(2):1–26, 2021.
5. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime: Reducing Attack Surface at Low Cost. In *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, volume 10719 of *Lecture Notes in Computer Science*, pages 235–260. Springer, 2017.
6. Daniel J Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. Ntru prime: round 3. *Submission to the NIST PQC standardization process, URl: https://ntruprime.cr.yp.to*, 2020.
7. Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):340–398, 2019.
8. Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking Kyber: First- and Higher-Order Implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):173–214, 2021.

9. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
10. Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John M Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. NTRU - Algorithm Specifications And Supporting Documentation. *Brown University and Onboard security company, Wilmington USA*, 2019.
11. Ming-Shing Chen and Tung Chou. Classic McEliece on the ARM Cortex-M4. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):125–148, 2021.
12. Ming-Shing Chen, Tim Güneysu, Markus Krausz, and Jan Philipp Thoma. Carry-less to BIKE faster. In *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, Lecture Notes in Computer Science, page to appear. Springer, 2022.
13. Chi-Ming Marvin Chung, Vincent Hwang, Matthias J. Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. NTT Multiplication for NTT-unfriendly Rings New Speed Records for Saber and NTRU on Cortex-M4 and AVX2. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):159–188, 2021.
14. Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order Polynomial Comparison and Masking Lattice-based Encryption. *Cryptology ePrint Archive*, 2021.
15. Jan-Pieter D'Anvers, Michiel Van Beirendonck, and Ingrid Verbauwhede. Revisiting Higher-Order Masked Comparison for Lattice-Based Cryptography: Algorithms and Bit-sliced Implementations. *IACR Cryptol. ePrint Arch.*, page 110, 2022.
16. A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards Sound and Optimal Leakage Detection Procedure. In *CARDIS*, volume 10728 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2017.
17. Nir Drucker, Shay Gueron, and Dusan Kostic. Fast Polynomial Inversion for Post Quantum QC-MDPC Cryptography. In *CSCML*, volume 12161 of *Lecture Notes in Computer Science*, pages 110–127. Springer, 2020.
18. Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):414–460, 2021.
19. Daniel Heinz, Matthias J. Kannwischer, Georg Land, Thomas Pöppelmann, Peter Schwabe, and Daan Sprenkels. First-Order Masked Kyber on ARM Cortex-M4. Cryptology ePrint Archive, Report 2022/058, 2022. `https://ia.cr/2022/058`.
20. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.
21. Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-Speed Key Encapsulation from NTRU. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 232–252. Springer, 2017.
22. Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
23. Toshiya Itoh and Shigeo Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in GF(2ˆm) Using Normal Bases. *Inf. Comput.*, 78(3):171–177, 1988.
24. Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. `https://github.com/mupq/pqm4`.

25. Emre Karabulut, Erdem Alkim, and Aydin Aysu. Single-Trace Side-Channel Attacks on ω-Small Polynomial Sampling: With Applications to NTRU, NTRU Prime, and CRYSTALS-DILITHIUM. In *HOST*, pages 35–45. IEEE, 2021.
26. Suparna Kundu, Jan-Pieter D'Anvers, Michiel Van Beirendonck, Angshuman Karmakar, and Ingrid Verbauwhede. Higher-order masked Saber. *IACR Cryptol. ePrint Arch.*, page 389, 2022.
27. Ching-Lin Li. Implementation of Polynomial Modular Inversion in Lattice based cryptography on ARM, 2021.
28. Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and IC Bourges. Hamming Quasi-Cyclic (HQC) – Third round version, 2021.
29. Victor S Miller. Use of Elliptic Curves in Cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.
30. Catinca Mujdei, Arthur Beckers, Jose Bermundo, Angshuman Karmakar, Lennert Wouters, and Ingrid Verbauwhede. Side-Channel Analysis of Lattice-Based Post-Quantum Cryptography: Exploiting Polynomial Multiplication. *IACR Cryptol. ePrint Arch.*, page 474, 2022.
31. Jan Richter-Brockmann, Ming-Shing Chen, Santosh Ghosh, and Tim Güneysu. Racing BIKE: Improved Polynomial Multiplication and Inversion in Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):557–588, 2022.
32. Jan Richter-Brockmann, Johannes Mono, and Tim Güneysu. Folding BIKE: Scalable Hardware Implementation for Reconfigurable Devices. *IEEE Trans. Computers*, 71(5):1204–1215, 2022.
33. Ronald L Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
34. Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.
35. Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, volume 11443 of *Lecture Notes in Computer Science*, pages 534–564. Springer, 2019.
36. Peter W Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM review*, 41(2):303–332, 1999.
37. Bo-Yeon Sim, Jihoon Kwon, Kyu Young Choi, Jihoon Cho, Aesun Park, and Dong-Guk Han. Novel Side-channel Attacks on Quasi-cyclic Code-based Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, pages 180–212, 2019.