IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 October 2023, accepted 22 December 2023, date of publication 1 January 2024,
date of current version 9 January 2024.

Digital Object Identifier 10.1109/ACCESS.2023.3348937

RESEARCH ARTICLE

Applying the Simple Partial Discard Method to Crystals-Kyber

DONGYOUNG ROH, (Member, IEEE), AND SANGIM JUNG, (Member, IEEE)
The Affiliated Institute of Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, Republic of Korea

Corresponding author: Dongyoung Roh (dyroh@nsr.re.kr)

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korean Government through Ministry of Science and ICT (MSIT) (Development of next-generation cryptosystem to improve security and usability of national information system) under Grant 2021-0-00046.

ABSTRACT In certain cryptographic applications random numbers are required (e.g., when generating cryptographic keys and generating digital signatures). To obtain these random numbers, the typical approach involves obtaining random bits first and then converting them into random numbers. Several methods to convert a sequence of random bits into a sequence of random numbers are known and some of them are standardized. Recently, ISO/IEC JTC 1/SC 27/WG 2 decided to add two more methods, the simple partial discard method and the complex partial discard method, to the existing four standard methods. Meanwhile, CRYSTALS-Kyber is the only public-key encryption and key-establishment algorithm selected for the post-quantum cryptography standardization project by NIST (National Institute of Standards and Technology). It uses an algorithm called Parse that takes a byte stream as input and outputs a polynomial of degree d with coefficients in $\mathbb{Z}_q$ (for some positive integer d and prime q) using the simple discard method to generate key pairs. In this paper, we apply the simple partial discard method to Parse. We show that using the simple partial discard method instead of the simple discard method can reduce the number of bits required by up to 12%. Furthermore, we show that, in some cases, using the simple partial discard method instead of the simple discard method can experimentally generate a polynomial up to 8% faster.

INDEX TERMS Conversion methods for random number generation, CRYSTALS-Kyber, simple discard method, simple partial discard method.

I. INTRODUCTION

Random number generation stands as a fundamental yet vital element within secure communication systems. This is due to the pivotal role that random numbers assume in cryptography, such as their use in generating cryptographic keys, initial vectors, primes, random exponents, and more. To generate random numbers, we typically start by using a random bit generator to produce a sufficient number of random bits, which are then converted into random numbers within the desired range. However, improper conversion methods can sometimes lead to security vulnerabilities of cryptographic algorithms. A notable example is the attack on Digital Signature Algorithm (DSA), a public-key cryptosystem and Federal Information Processing Standard for digital signatures [1]. DSA uses a randomly selected positive integer k, which is less than a certain n-bit prime q. The initial standard random generator in DSA simply set k as a n-bit random number reduced modulo q. Bleichenbacher exploited the non-uniformity of this random generator to expose weaknesses in DSA [2]. Therefore, we should be careful when converting random bits into random numbers.

Several conversion methods for random number generation have been developed and some of them have been standardized. There are four methods, the simple discard method, the complex discard method, the simple modular method, and the complex modular method, in ISO/IEC 18031:2011 [3] and ANSI X9.82-1-2006 (R2013) [4]. Three of them, the simple discard method, the complex discard method, and the simple modular method, are also in NIST SP 800-90A Rev. 1 [5]. Recently, the ISO/IEC JTC 1/SC 27/WG 2, a working group on cryptography and security mechanisms,

The associate editor coordinating the review of this manuscript and approving it for publication was Shuangqing Wei.

decided to introduce two additional conversion methods, the simple partial discard method and the complex partial discard method, in a new edition of ISO/IEC 18031.

Meanwhile, after it became known that widely used public-key cryptosystems are vulnerable to attacks using quantum computers, various cryptosystems have been developed to address this threat. Recently, the standardization of quantum-resistant cryptosystems has been progressing in several organizations. One of the most well-known efforts is the post-quantum cryptography standardization project by NIST (National Institute of Standards and Technology). Initiated in 2016, this project selected the primary algorithms for standardization in 2022 after receiving multiple proposals and undergoing several rounds of evaluation. The selected algorithms are categorized into public-key encryption and key-establishment algorithms and digital signature algorithms, with CRYSTALS-Kyber being the sole public-key encryption and key-establishment algorithm selected for standardization.

The key generation algorithm of CRYSTALS-Kyber requires a number of random numbers in $\mathbb{Z}_q$, where q is a prime. To generate these numbers, the designers of CRYSTALS-Kyber proposed an algorithm called Parse, a sub-algorithm of the key generation algorithm, which takes as input a sequence of bytes from the output of an extendable output function and iteratively generates random numbers in $\mathbb{Z}_q$. Algorithm Parse incorporates the simple discard method.

Our Contributions: In this paper, we propose applying the simple partial discard method, requiring fewer bits than the simple discard method, to Algorithm Parse. Specifically, we applied the simple partial discard method to an algorithm called Parse, which is used to generate a matrix $\hat{\mathbf{A}} \in R_q^{k\times k}$ in NTT (Number Theoretic Transform) domain in key generation. It is the simple discard method that the designers of CRYSTALS-Kyber uses in Algorithm Parse. We propose three algorithms that all use the simple partial discard method, but slightly differently. We theoretically analyze the performance and features of the proposed algorithms. We also experimentally compare the performance of the proposed algorithms with Algorithm Parse using the reference code provided by the designers of CRYSTALS-Kyber. We demonstrate that employing the simple partial discard method, as opposed to the simple discard method, can lead to a reduction in the required number of bits by up to 12%. Additionally, our experimental findings indicate that in certain scenarios, utilizing the simple partial discard method over the simple discard method can result in up to an 8% faster generation of the matrix $\hat{\mathbf{A}}$. And the generating function we obtain in the course of analyzing the proposed algorithms seems to be of independent interest.

Paper Organization: The outline of the paper is as follows. We provide notations and brief descriptions of conversion methods for random number generation and CRYSTALS-Kyber in Section II. Section III analyzes the characteristics of an algorithm called Parse designed by the designers of CRYSTALS-Kyber, a sub-algorithm of the key generation of CRYSTALS-Kyber. In Section IV, we propose three variations of the key generation algorithm of CRYSTALS-Kyber by applying the simple partial discard method and analyze the characteristics of them. Section V presents experimental results. Finally, Section VI concludes the paper.

II. PRELIMINARIES

A. NOTATIONS

We follow the notation of [6]. We denote by $\mathcal{B}$ the set $\{0, \ldots, 255\}$, i.e., the set of 8-bit unsigned integers (bytes). Consequently, we denote by $\mathcal{B}^k$ the set of byte arrays of length k and by $\mathcal{B}^*$ the set of byte arrays of arbitrary length (or byte streams).

We denote by R the ring $\mathbb{Z}[X]/(X^n+1)$ and by R_q the ring $\mathbb{Z}_q[X]/(X^n+1)$, where $n = 2^{n'-1}$ such that X^n+1 is the $2^{n'}$-th cyclotomic polynomial. Here and subsequently, the values of n, n', and q are fixed to $n = 256$, $n' = 9$, and $q = 3329$.

For any positive integer α, we define $r' = r \bmod^+ \alpha$ to be the unique element r' in the range $0 \le r' < \alpha$ such that $r' = r \bmod \alpha$.

Let $\mathrm{G} : \mathcal{B}^* \to \mathcal{B}^{32} \times \mathcal{B}^{32}$ be a hash function and $\mathsf{XOF} : \mathcal{B}^* \times \mathcal{B} \times \mathcal{B} \to \mathcal{B}^*$ an extendable output function. The designers of CRYSTALS-Kyber instantiated XOF with SHAKE-128 and G with SHA3-512 from the FIPS 202 standard [7]. They also proposed "90s" variant of CRYSTALS-Kyber in which they instantiated $\mathsf{XOF}(\rho, i, j)$ with AES-256 in CTR mode, where ρ is used as the key and $i\|j$ is zero-padded to a 12-byte nonce and the counter of CTR mode is initialized to zero and G with SHA-512.

B. CONVERSION METHODS FOR RANDOM NUMBER GENERATION

When randomness is needed, we typically employ random bit generators. However, certain cryptographic applications demand sequences of random numbers $(a_0, a_1, a_2, \cdots)$ where:

- for some positive integer r (the *range* of the random numbers), each integer a_i satisfies $0 \le a_i \le r-1$;
- for any $i \ge 0$ and s $(0 \le s \le r-1)$, $a_i = s$ with probability almost exactly $1/r$; and
- each integer a_i is statistically independent of any set of integers a_j $(j \ne i)$.

Several methods for converting a sequence of random bits into a sequence of random numbers are known, and among them, four methods have been standardized (see ISO/IEC 18031:2011 [3], NIST SP 800-90A Rev. 1 [5], and ANSI X9.82-1-2006 (R2013) [4]). The standardized methods consist of two types of discard methods, the simple discard method and the complex discard method, and two types of modular methods, the simple modular method and the complex modular method. These methods are simple and intuitive, but they also have drawbacks. The discard methods sometimes require discarding more bits than necessary, and the modular methods do not guarantee a uniform output.

Recently, the ISO/IEC JTC 1/SC 27/WG 2, responsible for standardizing cryptography and security mechanisms within ISO/IEC, decided to incorporate two new conversion methods for random number generation into the standard. These methods, developed by Koo et al. [8], are the simple partial discard method and the complex partial discard method, The partial discard methods discard less bits than discard methods and guarantee a uniform output.

Here, we briefly review the simple discard method and the simple partial discard method.

1) THE SIMPLE DISCARD METHOD

Let m be the unique positive integer satisfying $2^{m-1} < r \leq 2^m$ (m is uniquely defined by the choice of r). The method to generate the random number a is as follows.

1) Use the RBG to generate a sequence of m random bits, $(b_0, b_1, \ldots, b_{m-1})$.
2) Let $c = \sum_{i=0}^{m-1} 2^i b_i$.
3) If $c < r$ then put $a = c$, else discard c and go to step 1.

2) THE SIMPLE PARTIAL DISCARD METHOD

Let m be the unique positive integer satisfying $m^{m-1} < r \leq 2^m$ (m is uniquely defined by the choice of r). The method to generate the random number a is as follows.

1) Use the RBG to generate a sequence of m random bits, $(b_0, b_1, \ldots, b_{m-1})$.
2) Let $c = \sum_{i=0}^{m-1} 2^i b_i$.
3) If $c > r$ then put $a = c$, else
   a) let d be the unique integer such that $2^{m-d-1} \leq c \oplus (r-1) < 2^{m-d}$,
   b) use the RBG to generate a sequence of $d+1$ random bits, $(b_m, b_{m+1}, \ldots, b_{m+d})$,
   c) let $b_i = b_{m+i}$ for $0 \leq i < d+1$ and $b_{d+1+i} = b_i$ for $0 \leq i < m-d-1$ and go to step 2.

C. CRYSTALS-KYBER

CRYSTALS-Kyber is a quantum resistant IND-CCA2 (indistinguishability under adaptive chosen ciphertext attack) secure key encapsulation mechanism (KEM). Its security is based on the hardness of solving the learning-with-errors (LWE) problem over module lattices. In 2022, it was selected for the Post-Quantum Cryptography (PQC) standardization project by NIST. It has three different parameter sets for different security levels. Kyber512 is designed to be as secure as AES-128, Kyber768 is designed to be as secure as AES-192, and Kyber1024 is designed to be as secure as AES-256.

Recently, NIST published a draft standard FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard [9]. This standard specifies the algorithms and parameter sets of the ML-KEM scheme that is derived from the round-three version of the CRYSTALS-Kyber KEM [6]. Furthermore, ISO/IEC is also developing a standard for quantum resistant KEMs, including CRYSTALS-Kyber [10].

Algorithm 1 Kyber.**CPAPKE.KeyGen**(): Key Generation

Output: Secret key $sk \in \mathcal{B}^{12\cdot k\cdot n/8}$
Output: Public key $pk \in \mathcal{B}^{12\cdot k\cdot n/8+32}$
$d \leftarrow \mathcal{B}^{32}$
$(\rho, \sigma) := \mathrm{G}(d)$
$N := 0$
for i from 0 to $k-1$ **do**
  for j from 0 to $k-1$ **do**
    $\hat{\mathbf{A}}[i][j] := \mathbf{Parse}(\mathbf{XOF}(\rho, j, i))$
  end for
end for
...

For the detailed specifications of CRYSTALS-Kyber we refer the reader to [6] and [9].

III. THE KEY GENERATION OF CRYSTALS-KYBER

The key generation algorithm of CRYSTALS-Kyber outputs a secret key $sk \in \mathcal{B}^{12\cdot k\cdot n/8}$ a public key $pk \in \mathcal{B}^{12\cdot k\cdot n/8+32}$. To output a public key, the key generation algorithm first generates a matrix $\hat{\mathbf{A}} \in R_q^{k\times k}$ in NTT (Number Theoretic Transform) domain. The matrix is generated by repeatedly invoking an algorithm called Parse k^2 times. Algorithm Parse takes a byte stream $B = b_0, b_1, b_2 \cdots \in \mathcal{B}^*$ and outputs the NTT-representation $\hat{a} = \hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1}X^{n-1} \in R_q$ of $a \in R_q$ The input byte stream of Algorithm Parse is computed using XOF. The function XOF is recommended to be instantiated with SHAKE-128 or AES-256 in CTR mode.

To uniformly sample an element in R_q, Algorithm Parse takes a chunk of twelve bits at a time from the input byte stream and sets a coefficient of the element only when the chunk is in the desired range. If the chunk is not in the range, then it takes another chunk of twelve bits (the simple discard method). It repeats this process until it has determined all the coefficients of the element in R_q. So the number of bytes (or bits) required by Algorithm Parse to output $\hat{a} \in R_q$ is not fixed. It varies depending on the input byte streams.

Let X_0 be a random variable that represents the number of bits for Algorithm Parse to compute a coefficient in $\mathbb{Z}_q$. It is easy to see that the expected value of X_0 is $\mathbf{E}[X_0] = 12 \times \frac{4096}{3329} \approx 14.76479$. Then the expected number of bits required by Algorithm Parse to output an element in R_q is $256 \times \mathbf{E}[X_0] \approx 3779.787$.

First, suppose that the function XOF that outputs the input byte stream of Algorithm Parse is instantiated with SHAKE-128. Since SHAKE-128 outputs 1,344 bits per squeezing step, the minimum required number of squeezing steps is $\lceil (12 \times 256)/1344 \rceil = 3$ and the expected number of squeezing steps is $\lceil 256 \times \mathbf{E}[X_0]/1344 \rceil = 3$. So, the reference code, provided by the designers of CRYSTALS-Kyber, sets three as a default number of the squeezing steps. This means that when preparing the input byte stream for Algorithm Parse, the squeezing step of SHAKE-128 is called three

Algorithm 2 Parse: $\mathcal{B}^* \to R_q$

Input: Byte stream $B = b_0, b_1, b_2 \cdots \in \mathcal{B}^*$
Output: NTT-representation $\hat{a} \in R_q$ of $a \in R_q$

```
i := 0
j := 0
while j < n do
    d1 := b_i + 256 · (b_{i+1} mod+ 16)
    d2 := ⌊b_{i+1}/16⌋ + 16 · b_{i+2}
    if d1 < q then
        â_j := d1
        j := j + 1
    end if
    if d2 < qandj < n then
        â_j := d2
        j := j + 1
    end if
    i := i + 3
end while
return â0 + â1 X + ··· + â_{n−1} X^{n−1}
```

times to prepare the first 4,032 bits. If it is unable to successfully output an element in R_q using the 4,032 bits, an additional squeezing step is called one at a time until it is able to do so. However, three squeezing steps are not always sufficient. We can easily see that the probability that Algorithm Parse requires more than three squeezing steps is

$$\sum_{i=0}^{255} \binom{336}{i} \left(\frac{3329}{4096}\right)^i \left(1 - \frac{3329}{4096}\right)^{336-i}.$$

To compute it, we recall a well-known fact that the cumulative distribution function $F(k; m, p)$ for a binomial distribution $X \sim B(m, p)$ can be computed as follows.

$$\begin{aligned} F(k; m, p) &= \Pr(X \le k) \\ &= \sum_{i=0}^{\lfloor k \rfloor} \binom{m}{i} p^i (1-p)^{m-i} \\ &= (m-k)\binom{m}{k} \int_0^{1-p} t^{m-k-1}(1-t)^k dt \end{aligned} \quad (1)$$

Therefore,

$$\begin{aligned} &\sum_{i=0}^{255} \binom{336}{i} \left(\frac{3329}{4096}\right)^i \left(1 - \frac{3329}{4096}\right)^{336-i} \\ &= F\left(255; 336, \frac{3329}{4096}\right) \\ &= (336-255)\binom{336}{255} \int_0^{1-\frac{3329}{4096}} t^{336-255-1}(1-t)^{255} dt \\ &\approx 0.008328 \approx 2^{-6.9079}. \end{aligned}$$

Next, suppose that the function XOF that outputs the input byte stream of Algorithm Parse is instantiated with AES-256 in CTR mode. Since AES-256 in CTR mode outputs 128 bits per invocation to AES-256, the minimum number and the expected number of invocations of AES-256 are $\lceil (12 \times 256)/128 \rceil = 24$ and $\lceil 256 \times \mathbf{E}[X_0]/128 \rceil = 30$, respectively. The reference code, provided by the designers of CRYSTALS-Kyber, sets 32 as a default number of the invocations. This means that when preparing the input byte stream for Algorithm Parse, AES-256 is called thirty-two times to prepare the first 4,096 bits. If it does not succeed in using those 4,096 bits to output an element in R_q, the reference code calls AES-256 four more times, one at a time, until it does. By using (1), we can see that the probability that Algorithm Parse requires more than 32 invocations of AES-256 is

$$\begin{aligned} &\sum_{i=0}^{255} \binom{341}{i} \left(\frac{3329}{4096}\right)^i \left(1 - \frac{3329}{4096}\right)^{341-i} \\ &= F\left(255; 341, \frac{3329}{4096}\right) \approx 0.001836 \approx 2^{-9.0890}. \end{aligned}$$

And we can also see that the probability that Algorithm Parse requires more than 28 invocations of AES-256 is

$$\begin{aligned} &\sum_{i=0}^{255} \binom{298}{i} \left(\frac{3329}{4096}\right)^i \left(1 - \frac{3329}{4096}\right)^{298-i} \\ &= F\left(255; 298, \frac{3329}{4096}\right) \approx 0.97866 \approx 2^{-0.031123}. \end{aligned}$$

IV. APPLYING THE SIMPLE PARTIAL DISCARD METHOD TO THE KEY GENERATION OF CRYSTALS-KYBER

In this section, we apply the simple partial discard method to the key generation algorithm of CRYSTALS-Kyber. According to the specifications of CRYSTALS-Kyber, Algorithm Parse generates polynomials in R_q using the simple discard method during the key generation. Here, we propose modified algorithms where the polynomials are generated using the simple partial discard method instead of the simple discard method.

Specifically, we propose three algorithms. The first algorithm employs the simple partial discard method without any modifications, while the second and third algorithms employ modified versions of the simple partial discard method. The reasons why we propose three different (but similar) algorithms are as follows. The simple partial discard method offers an advantage over the simple discard method by utilizing fewer random bits while outputting the same number of random numbers. However, achieving this advantage requires more computations and/or time to convert a sequence of random bits into a sequence of random numbers for the simple partial discard method compared to the simple discard method. In other words, there exists a trade-off relationship between the simple discard method and the simple partial discard method. If generating the input random bits requires minimal computations and/or time, it would be advantageous to use the simple discard method, which requires less computations and/or time. On the other hand, if generating the input random bits requires more time than additional computations and/or time, then using the simple

Algorithm 3 **Parse_SPDM1**: $\mathcal{B}^* \to R_q$

Input: Byte stream $B = b_0, b_1, b_2 \cdots \in \mathcal{B}^*$
Output: NTT-representation $\hat{a} \in R_q$ of $a \in R_q$

$i := 0$
$j := 0$
$k := 0$
while $j < n$ **do**
  if $0 \le k < 5$ **then**
    $d := \left(2^{k+4} \cdot b_i \bmod^+ 2^{12}\right) + \lfloor b_{i+1}/2^{4-k} \rfloor$
  else
    $d := \left(2^{k+4} \cdot b_i \bmod^+ 2^{12}\right) + 2^{k-1} \cdot b_{i+1} + \lfloor b_{i+2}/2^{12-k} \rfloor$
  end if
  if $d < q$ **then**
    $\hat{a}_j := d$
    $i := i + \lfloor k/4 \rfloor + 1$
    $j := j + 1$
    $k := (k+4) \bmod^+ 8$
  else
    $i := \lfloor \left(\mathbf{spdm}[d] + k\right)/8 \rfloor$
    $k := \left(\mathbf{spdm}[d] + k\right) \bmod^+ 8$
  end if
end while
return $\hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1} X^{n-1}$

partial discard method would be favorable. The second and third variations leverage this trade-off between computational complexity and the number of random bits required. Among the three modified algorithms, the first requires the fewest random bits and the most amount of computations and/or time, while the third requires the most random bits and uses the least amount of computations and/or time.

A. THE FIRST ALGORITHM

In this subsection, we propose and analyze the first modified algorithm Parse_SPDM1. As mentioned earlier, Algorithm Parse_SPDM1 adopts the simple partial discard method without any modifications.

Here, the table spdm in Parse_SPDM1 is an array of 4,096 numbers indicating the number of discarded bits. That is, $\mathsf{spdm}[i] = 0$ for $0 \le i < 3329$ and $\mathsf{spdm}[i] = x$ for $3329 \le i < 4096$, where x is an integer such that $2^{12-x} \le (i \oplus 3328) < 2^{11-x}$. The concrete values of the table spdm is given in Table 1.

Let X_1 be a random variable that represents the number of bits for Algorithm Parse_SPDM1 to compute a coefficient in $\mathbb{Z}_q$. The expected value of X_1 satisfies

$$\begin{aligned}\mathbf{E}[X_1] = 12 \times \frac{3329}{4096} &+ (\mathbf{E}[X_1] + 12) \times \frac{1}{4096} \\ &+ (\mathbf{E}[X_1] + 11) \times \frac{2}{4096} + (\mathbf{E}[X_1] + 10) \times \frac{4}{4096} \\ &+ (\mathbf{E}[X_1] + 9) \times \frac{8}{4096} + (\mathbf{E}[X_1] + 8) \times \frac{16}{4096} \\ &+ (\mathbf{E}[X_1] + 7) \times \frac{32}{4096} + (\mathbf{E}[X_1] + 6) \times \frac{64}{4096} \\ &+ (\mathbf{E}[X_1] + 5) \times \frac{128}{4096} + (\mathbf{E}[X_1] + 3) \times \frac{512}{4096}.\end{aligned}$$

Therefore, $\mathbf{E}[X_1] = 43006/3329 \approx 12.91859$. Then the expected number of bits needed by Algorithm Parse_SPDM1 to output an element in R_q is $256 \times \mathbf{E}[X_1] \approx 3307.16$.

TABLE 1. The table spdm.

i	$\mathsf{spdm}[i]$
$0 \le i < 3329$	0
$3329 \le i < 3330$	12
$3330 \le i < 3332$	11
$3332 \le i < 3336$	10
$3336 \le i < 3344$	9
$3344 \le i < 3360$	8
$3360 \le i < 3392$	7
$3392 \le i < 3456$	6
$3456 \le i < 3584$	5
$3584 \le i < 4096$	3

First, suppose that the function XOF that outputs the input byte stream of Algorithm Parse_SPDM1 is instantiated with SHAKE-128. The minimum required number of squeezing steps is $\lceil (12 \times 256)/1344 \rceil = 3$ and the expected number of squeezing steps is $\lceil 256 \times \mathbf{E}[X_1]/1344 \rceil = 3$. The minimum required number and the expected number of squeezing steps of Algorithm Parse_SPDM1 are the same as those of Algorithm Parse.

However, the probability that Algorithm Parse_SPDM1 requires more than three squeezing steps is different from that Algorithm Parse requires. Now we compute the probability that Algorithm Parse_SPDM1 requires more than three squeezing steps. Let $G_1(x, y)$ be the generating function of $F_1(\alpha, \beta)$, the probability that Algorithm Parse_SPDM1 outputs β coefficients in $\mathbb{Z}_q$ for given α random bits, where α and β are non-negative integers. Then the probability that Algorithm Parse_SPDM1 requires more than three squeezing steps is

$$\sum_{i=0}^{255} \frac{\frac{\partial G_1(x,y)}{\partial x^{4032} \partial y^i}(0,0)}{4032! \cdot i!} \approx 2^{-144.0427259}$$

by Proposition 1 in Appendix.

Next, suppose that the function XOF that outputs the input byte stream of Algorithm Parse_SPDM1 is instantiated with AES-256 in CTR mode. The minimum number and the expected number of invocations of AES-256 are $\lceil (12 \times 256)/128 \rceil = 24$ and $\lceil 256 \times \mathbf{E}[X_1]/128 \rceil = 26$, respectively. The probability that Algorithm Parse_SPDM1 requires more than t invocations of AES-256 is

$$\sum_{i=0}^{255} \frac{\frac{\partial G_1(x,y)}{\partial x^{128t} \partial y^i}(0,0)}{(128t)! \cdot i!}.$$

The probabilities for several values of t are given in Table 2.

TABLE 2. The probability that Algorithm Parse_SPDM1 requires more than t invocations of AES-256.

t	Probability
32	$2^{-163.1331964}$
31	$2^{-125.6020613}$
30	$2^{-90.95414030}$
29	$2^{-59.89899649}$
28	$2^{-33.47476964}$
27	$2^{-13.29169597}$
26	$2^{-1.888114929}$

Algorithm 4 Parse_SPDM2: $\mathcal{B}^* \to R_q$

Input: Byte stream $B = b_0, b_1, b_2 \cdots \in \mathcal{B}^*$
Output: NTT-representation $\hat{a} \in R_q$ of $a \in R_q$

```
i := 0
j := 0
k := 0
while j < n do
    if k = 0 then
        d := 16 · b_i + ⌊b_{i+1}/16⌋
    else
        d := (256 · b_i mod 2^12) + b_{i+1}
    end if
    if d < 3329 then
        â_j := d
        i := i + k + 1
        j := j + 1
        k := k ⊕ 1
    else if 3329 ≤ d < 3344 then
        i := i + k + 1
        k := k ⊕ 1
    else if 3344 ≤ d < 3584 then
        i := i + 1
    else
        i := i + k
        k := k ⊕ 1
    end if
end while
return â_0 + â_1 X + ··· + â_{n−1} X^{n−1}
```

B. THE SECOND ALGORITHM

In this subsection, we propose and analyze the second modified algorithm Parse_SPDM2. As mentioned earlier, Algorithm Parse_SPDM2 adopts the simple partial discard method in a slightly modified form to reduce computations and/or time. To achieve this, it requires more random bits than Algorithm Parse_SPDM1, but it still requires fewer random bits than Algorithm Parse.

Let X_2 be a random variable that represents the number of bits for Algorithm Parse_SPDM2 to compute a coefficient in $\mathbb{Z}_q$. The expected value of X_2 satisfies

$$\mathbf{E}[X_2] = 12 \times \frac{3329}{4096} + (\mathbf{E}[X_2] + 12) \times \frac{15}{4096} + (\mathbf{E}[X_1] + 8) \times \frac{240}{4096} + (\mathbf{E}[X_1] + 4) \times \frac{512}{4096}.$$

TABLE 3. The probability that Algorithm Parse_SPDM2 requires more than t invocations of AES-256.

t	Probability
32	$2^{-89.54777354}$
31	$2^{-65.99860662}$
30	$2^{-44.93048123}$
29	$2^{-26.90896429}$
28	$2^{-12.72875244}$
27	$2^{-3.499014167}$

Therefore, $\mathbf{E}[X_2] = 44096/3329 \approx 13.24602$. Then the expected number of bits required by Algorithm Parse_SPDM2 to output an element in R_q is $256 \times \mathbf{E}[X_2] \approx 3390.98108$.

First, suppose that the function XOF that outputs the input byte stream of Algorithm Parse_SPDM2 is instantiated with SHAKE-128. The minimum required number of squeezing steps is $\lceil (12 \times 256)/1344 \rceil = 3$ and the expected number of squeezing steps is $\lceil 256 \times \mathbf{E}[X_2]/1344 \rceil = 3$. The minimum required number and the expected number of squeezing steps of Algorithm Parse_SPDM2 are the same as those for Algorithm Parse and Algorithm Parse_SPDM1.

However, the probability that Algorithm Parse_SPDM2 requires more than three squeezing steps differs from the probabilities that Algorithm Parse and Algorithm Parse_SPDM1 require. Now we compute the probability that Algorithm Parse_SPDM2 requires more than three squeezing steps. Let $G_2(x, y)$ be the generating function of $F_2(\alpha, \beta)$, the probability that Algorithm Parse_SPDM2 outputs β coefficients in $\mathbb{Z}_q$ for given α random bits, where α and β are non-negative integers. Then the probability that Algorithm Parse_SPDM2 requires more than three squeezing steps is

$$\sum_{i=0}^{255} \frac{\frac{\partial G_2(x,y)}{\partial x^{4032} \partial y^i}(0,0)}{4032! \cdot i!} \approx 2^{-77.49183682}$$

by Proposition 2 in Appendix.

Next, suppose that the function XOF that outputs the input byte stream of Algorithm Parse_SPDM2 is instantiated with AES-256 in CTR mode. The minimum number and the expected number of invocations of AES-256 are $\lceil (12 \times 256)/128 \rceil = 24$ and $\lceil 256 \times \mathbf{E}[X_2]/128 \rceil = 27$, respectively. The probability that Algorithm Parse_SPDM2 requires more than t invocations of AES-256 is

$$\sum_{i=0}^{255} \frac{\frac{\partial G_2(x,y)}{\partial x^{128t} \partial y^i}(0,0)}{(128t)! \cdot i!}.$$

The probabilities for several values of t are given in Table 3.

C. THE THIRD ALGORITHM

In this subsection, we propose and analyze the third modified algorithm Parse_SPDM3. As mentioned earlier, Algorithm Parse_SPDM3 adopts the simple partial discard method in a slightly modified form to further reduce computations

Algorithm 5 Parse_SPDM3: $\mathcal{B}^* \to R_q$

Input: Byte stream $B = b_0, b_1, b_2 \cdots \in \mathcal{B}^*$
Output: NTT-representation $\hat{a} \in R_q$ of $a \in R_q$

```
i := 0
j := 0
while j < n do
    d1 := 16 · b_i + ⌊b_{i+1}/16⌋
    if d1 < 3584 then
        i := i + 1
    end if
    d2 := (256 · b_i mod+ 2^12) + b_{i+1}
    if d1 < q then
        â_j = d1
        j := j + 1
    end if
    if d2 < qandj < n then
        â_j = d2
        j := j + 1
    end if
    if d2 < 3584 then
        i := i + 2
    else
        i := i + 1
    end if
end while
return â_0 + â_1 X + ··· + â_{n−1} X^{n−1}
```

and/or time. While it requires more random bits than Algorithm Parse_SPDM1 and Algorithm Parse_SPDM2, it still requires fewer random bits than Algorithm Parse.

Let X_3 be a random variable that represents the number of bits for Algorithm Parse_SPDM3 to compute a coefficient in $\mathbb{Z}_q$. The expected value of X_3 satisfies

$$\mathbf{E}[X_3] = 12 \times \frac{3329}{4096} + (\mathbf{E}[X_3] + 12) \times \frac{255}{4096} + (\mathbf{E}[X_3] + 4) \times \frac{512}{4096}.$$

Therefore, $\mathbf{E}[X_3] = 45056/3329 \approx 13.53439$. Then the expected number of bits required by Algorithm Parse_SPDM3 to output an element in R_q is $256 \times \mathbf{E}[X_3] \approx 3464.80505$.

First, suppose that the function XOF that outputs the input byte stream of Algorithm Parse_SPDM3 is instantiated with SHAKE-128. The minimum required number of squeezing steps is $\lceil (12 \times 256)/1344 \rceil = 3$ and the expected number of squeezing steps is $\lceil 256 \times \mathbf{E}[X_3]/1344 \rceil = 3$. The minimum required number and the expected number of squeezing steps of Algorithm Parse_SPDM3 are the same as those of Algorithm Parse, those of Algorithm Parse_SPDM1, and those of Algorithm Parse_SPDM2.

However, the probability that Algorithm Parse_SPDM3 requires more than three squeezing steps is different from that Algorithm Parse requires, that Algorithm Parse_SPDM1 requires, and that Algorithm Parse_SPDM2 requires. Now we calculate the probability that Algorithm Parse_SPDM3 requires more than three squeezing steps. Let $G_3(x, y)$ be the generating function of $F_3(\alpha, \beta)$, the probability that Algorithm Parse_SPDM3 outputs β coefficients in $\mathbb{Z}_q$ for given α random bits, where α and β are non-negative integers. Then the probability that Algorithm Parse_SPDM3 requires more than three squeezing steps is

$$\sum_{i=0}^{255} \frac{\frac{\partial G_3(x,y)}{\partial x^{4032} \partial y^i}(0,0)}{4032! \cdot i!} \approx 2^{-41.71623662}$$

by Proposition 3 in Appendix.

Next, suppose that the function XOF that outputs the input byte stream of Algorithm Parse_SPDM3 is instantiated with AES-256 in CTR mode. The minimum number and the expected number of invocations of AES-256 are $\lceil (12 \times 256)/128 \rceil = 24$ and $\lceil 256 \times \mathbf{E}[X_3]/128 \rceil = 28$, respectively. The probability that Algorithm Parse_SPDM3 requires more than t invocations of AES-256 is

$$\sum_{i=0}^{255} \frac{\frac{\partial G_3(x,y)}{\partial x^{128t} \partial y^i}(0,0)}{(128t)! \cdot i!}.$$

The probabilities for several values of t are given in Table 4.

TABLE 4. The probability that Algorithm Parse_SPDM3 requires more than t invocations of AES-256.

t	Probability
32	$2^{-49.10127602}$
31	$2^{-34.76594383}$
30	$2^{-22.33360303}$
29	$2^{-12.19803860}$
28	$2^{-4.875442116}$

V. EXPERIMENTAL RESULTS

This section demonstrates experimental results of applying the simple partial discard method to the key generation of CRYSTALS-Kyber. We compare the performances of generating a matrix $\hat{\mathbf{A}}$ during the key generation of CRYSTALS-Kyber using Algorithms Parse, Parse_SPDM1, Parse_SPDM2, and Parse_SPDM3. We used the reference implementation by the designers of CRYSTALS-Kyber when measuring the performance of generating $\hat{\mathbf{A}}$ using Algorithm Parse and made minimal modifications to the reference implementation when measuring performances of generating $\hat{\mathbf{A}}$ using Algorithms Parse_SPDM1, Parse_SPDM2, and Parse_SPDM3.

All benchmarks were conducted on a single core of an AMD Ryzen™ Threadripper™ PRO 5995WX processor running at a fixed 2.7 GHz base clock, with TurboBoost and hyperthreading disabled. The benchmarking machine is equipped with 382 GB of RAM and runs Ubuntu 22.04.3 LTS. All implementations were compiled using gcc 11.4.0. The reported cycle counts and numbers of required bits represent the averages from one million executions of the respective function. Similarly, any reported probabilities of exceeding a predetermined (default) number of calls to the SHAKE-128

TABLE 5. Average cycle counts to generate the matrix $\hat{\mathbf{A}}$ when XOF is instantiated with SHAKE-128.

Cipher	Parse	Parse_SPDM1	Parse_SPDM2	Parse_SPDM3
Kyber512	24,218	31,597	27,763	26,109
Kyber768	54,351	70,598	62,920	58,494
Kyber1024	95,669	125,708	115,044	104,623

TABLE 6. Average numbers of bits required to generate an element in R_q when XOF is instantiated with SHAKE-128.

Cipher	Parse	Parse_SPDM1	Parse_SPDM2	Parse_SPDM3
Kyber512	3785.84464	3303.68821	3388.99038	3470.35001
Kyber768	3785.79937	3303.65490	3389.02135	3470.31770
Kyber1024	3785.82694	3303.64565	3388.95880	3470.32228

TABLE 7. Probabilities that the squeezing step of SHAKE-128 is required more than three times to output an element in R_q when generating $\hat{\mathbf{A}}$ one million times.

Cipher	Parse	Parse_SPDM1	Parse_SPDM2	Parse_SPDM3
Kyber512	$2^{-6.9125}$	0	0	0
Kyber768	$2^{-6.9112}$	0	0	0
Kyber1024	$2^{-6.9041}$	0	0	0

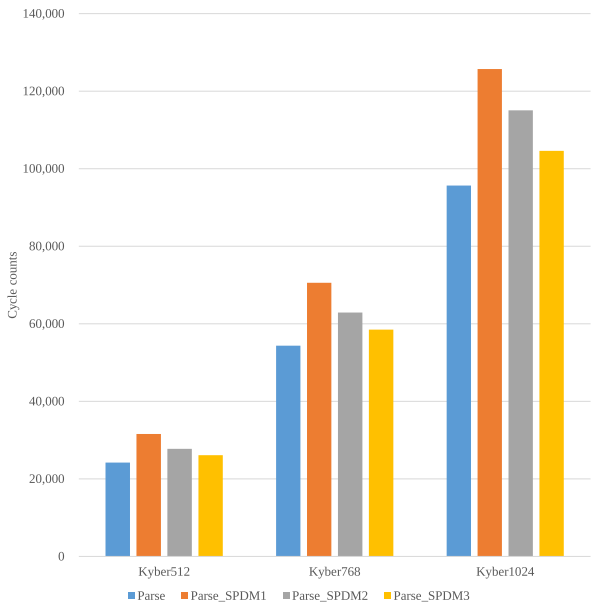

FIGURE 1. Average cycle counts to generate the matrix $\hat{\mathbf{A}}$ when XOF is instantiated with SHAKE-128.

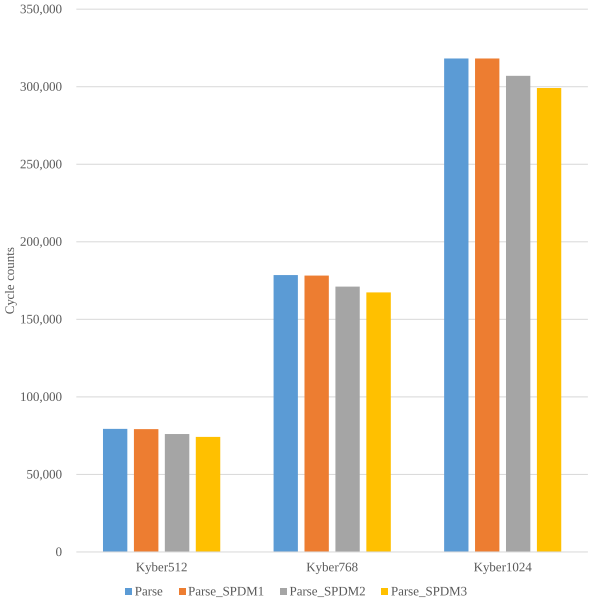

FIGURE 2. Average cycle counts to generate the matrix $\hat{\mathbf{A}}$ when XOF is instantiated with AES-256 in CTR mode with the default number of invocations of AES-256 fixed as 28.

squeezing step or AES-256 are based on one million executions of the respective function.

We first present the experimental results when XOF is instantiated with SHAKE-128. Table 5 and Fig. 1 show the average cycle counts to generate the matrix $\hat{\mathbf{A}}$ in the key generation of CRYSTALS-Kyber. Tables 6 and 7 show the average numbers of bits required and the probabilities of calling the squeezing step of SHAKE-128 more than three times to output an element in R_q, respectively. Algorithms Parse_SPDM1, Parse_SPDM2, Parse_SPDM3, and Parse required fewer bits, in that order, and these figures are roughly the same as the theoretical calculations in Section IV. The average number of bits is naturally independent of CRYSTALS-Kyber parameter sets. In addition, the probability that Algorithm Parse requires more than three squeezing steps was also the same as calculated theoretically. And Algorithms Parse_SPDM1, Parse_SPDM2, and Parse_SPDM3 never required more than three squeezing steps, as was expected. However, the number of bits required by the

TABLE 8. Average cycle counts to generate the matrix $\hat{\mathbf{A}}$ when XOF is instantiated with AES-256 in CTR mode.

Cipher	Parse		Parse_SPDM1		Parse_SPDM2		Parse_SPDM3	
	32^1	28^1	32^1	28^1	32^1	28^1	32^1	28^1
Kyber512-90s	81,223	79,418	87,289	79,245	83,725	76,114	81,999	74,277
Kyber768-90s	179,518	178,556	195,426	178,227	189,286	171,111	184,458	167,400
Kyber1024-90s	319,159	318,213	348,867	318,195	335,394	307,053	328,779	299,133

[1] The numbers "32" and "28" denote the default numbers of invocations of AES-256.

TABLE 9. Average numbers of bits required to generate an element in R_q when XOF is instantiated with AES-256 in CTR mode.

Cipher	Parse		Parse_SPDM1		Parse_SPDM2		Parse_SPDM3	
	32^1	28^1	32^1	28^1	32^1	28^1	32^1	28^1
Kyber512-90s	3785.71501	3785.76386	3303.66387	3303.64866	3388.96396	3388.94946	3470.28374	3470.26683
Kyber768-90s	3785.75283	3785.78535	3303.66154	3303.67946	3388.96962	3388.97487	3470.24236	3470.30215
Kyber1024-90s	3785.77158	3785.79901	3303.65715	3303.65792	3388.98885	3388.97035	3470.31045	3470.28392

[1] The numbers "32" and "28" denote the default numbers of invocations of AES-256.

TABLE 10. Probabilities that the invocation of AES-256 in CTR mode is required more than 32 or 28 times to output an element in R_q when generating $\hat{\mathbf{A}}$ one million times.

Cipher	Parse		Parse_SPDM1		Parse_SPDM2		Parse_SPDM3	
	32^1	28^1	32^1	28^1	32^1	28^1	32^1	28^1
Kyber512-90s	$2^{-8.6125}$	$2^{-0.031137}$	0	0	0	$2^{-12.747}$	0	$2^{-4.6620}$
Kyber768-90s	$2^{-8.6400}$	$2^{-0.031089}$	0	0	0	$2^{-12.702}$	0	$2^{-4.6591}$
Kyber1024-90s	$2^{-8.6305}$	$2^{-0.031143}$	0	0	0	$2^{-12.697}$	0	$2^{-4.6605}$

[1] The numbers "32" and "28" denote the default numbers of invocations of AES-256.

algorithm and its speed are inversely proportional. That is, Parse, which theoretically requires the most bits, was the fastest, and Parse_SPDM1, which theoretically requires the fewest bits, was the slowest.

Next, we present the experimental results when XOF is instantiated with AES-256 in CTR mode. Table 8 and Fig. 2 show the average cycle counts to generate the matrix $\hat{\mathbf{A}}$ in the key generation of CRYSTALS-Kyber. Tables 9 and 10 show the average numbers of bits required and the probabilities of calling AES-256 more than 32 and 28 times to output an element in R_q, respectively. Algorithms Parse_SPDM1, Parse_SPDM2, Parse_SPDM3, and Parse required fewer bits, in that order, and these figures are roughly the same as the theoretical calculations in Section IV. As earlier, the average number of bits is naturally independent of CRYSTALS-Kyber parameter sets. And they are independent of the algorithm that instantiates XOF. Not only are they independent of the algorithm that instantiates XOF, but they are also independent of the default number of iterations of AES-256 set by the implementation. Therefore, the numbers shown in Tables 6 and 9 are nearly identical. The probabilities of requiring more than 32 or 28 calls of AES-256 are the same as theoretically calculated. In particular, in some cases, it did not happen to call AES-256 more than the default number of times as was expected.

VI. CONCLUSION

In this paper, we applied the simple partial discard method to CRYSTALS-Kyber. Specifically, we applied the simple partial discard method to an algorithm called Parse, which is used to generate a matrix $\hat{\mathbf{A}} \in R_q^{k\times k}$ in NTT (Number Theoretic Transform) domain in key generation. The reference code provided by the designers of CRYSTALS-Kyber uses the simple discard method. We applied the simple partial discard method, which requires fewer bits than the simple discard method, and analyzed its impact theoretically and experimentally. As expected, our theoretical analysis and experimental results confirm that the utilization of the simple partial discard method requires fewer bits than that of the simple discard method. In addition, we experimentally verified that in certain cases, the simple partial discard method not only uses fewer bits, but also generates $\hat{\mathbf{A}}$ faster.

However, this research is only applicable to implementations that utilize a single core. While the simple discard method is highly intuitively parallelizable, the same ease of parallelization does not apply to the simple partial discard method. Therefore, the challenge of parallelizing the simple partial discard method and integrating it into cryptographic algorithms such as CRYSTALS-Kyber remains an open problem.

APPENDIX
GENERATING FUNCTIONS

In this appendix, we compute generating functions that can be used to calculate the probabilities that Algorithms Parse, Parse_SPDM1, Parse_SPDM2, and Parse_SPDM3 require more than predefined (default) number of calls to the SHAKE-128 squeezing step or AES-256.

First we compute the generating function of the probability $F_1(\alpha, \beta)$ that Algorithm Parse_SPDM1 outputs β coefficients in $\mathbb{Z}_q$ for given α random bits.

Proposition 1: Let $F_1(\alpha, \beta)$ be the probability that Algorithm *Parse_SPDM1* outputs β coefficients in $\mathbb{Z}_q$ for given α random bits, where α and β are non-negative integers. We assume that the algorithm tries to output a coefficient only when more than or equal to twelve random bits are left. And let $G_1(x, y)$ be the generating function of $F_1(\alpha, \beta)$. Then

$$G_1(x, y) = \frac{\sum_{i=5}^{12} f_{12-i}x^i + f_9x^3 - 1}{syx^{12} + \sum_{i=5}^{12} f_{12-i}x^i + f_9x^3 - 1} H_1(x),$$

where $s = 3329/4096$, $f_0 = 1/4096$, $f_1 = 2/4096$, $f_2 = 4/4096$, $f_3 = 8/4096$, $f_4 = 16/4096$, $f_5 = 32/4096$, $f_6 = 64/4096$, $f_7 = 128/4096$, $f_9 = 512/4096$, and

$$H_1(x) = \frac{\sum_{j=5}^{11}\sum_{i=12-j}^{7} f_i x^j + \sum_{i=3}^{11} f_9 x^i - \sum_{i=0}^{11} x^i}{\sum_{i=5}^{12} f_{12-i}x^i + f_9x^3 - 1}.$$

Proof: The probability $F_1(\alpha, \beta)$ satisfies the following recurrence relation.

$$\begin{aligned}
&F_1(\alpha + 12, \beta + 1)\\
&= sF_1(\alpha, \beta) + f_0F_1(\alpha, \beta + 1) + f_1F_1(\alpha + 1, \beta + 1)\\
&\quad + f_2F_1(\alpha + 2, \beta + 1) + f_3F_1(\alpha + 3, \beta + 1)\\
&\quad + f_4F_1(\alpha + 4, \beta + 1) + f_5F_1(\alpha + 5, \beta + 1)\\
&\quad + f_6F_1(\alpha + 6, \beta + 1) + f_7F_1(\alpha + 7, \beta + 1)\\
&\quad + f_9F_1(\alpha + 9, \beta + 1)
\end{aligned}$$

Here, $F_1(i, 0) = 1$ for $0 \le i < 12$ and $F_1(i, j) = 0$ for $0 \le i < 12$ and $1 \le j$. These are the boundary conditions.

Before computing the generating function of $F_1(\alpha, \beta)$, we first compute the generating function of $F_1(\alpha, 0)$. The probability $F_1(\alpha, 0)$ satisfies the following recurrence relation.

$$\begin{aligned}
&F_1(\alpha + 12, 0)\\
&= f_0F_1(\alpha, 0) + f_1F_1(\alpha + 1, 0) + f_2F_1(\alpha + 2, 0)\\
&\quad + f_3F_1(\alpha + 3, 0) + f_4F_1(\alpha + 4, 0) + f_5F_1(\alpha + 5, 0)\\
&\quad + f_6F_1(\alpha + 6, 0) + f_7F_1(\alpha + 7, 0) + f_9F_1(\alpha + 9, 0)
\end{aligned}$$

Let $H_1(x) = \sum_{i\ge0} F_1(i, 0)x^i$ be the generating function of $F_1(\alpha, 0)$. From the recurrence relation of $F_1(\alpha, 0)$, we obtain

$$\begin{aligned}
&\sum_{i\ge0} F_1(i + 12, 0)x^i\\
&= f_0\sum_{i\ge0} F_1(i, 0)x^i + f_1\sum_{i\ge0} F_1(i + 1, 0)x^i\\
&\quad + f_2\sum_{i\ge0} F_1(i + 2, 0)x^i + f_3\sum_{i\ge0} F_1(i + 3, 0)x^i\\
&\quad + f_4\sum_{i\ge0} F_1(i + 4, 0)x^i + f_5\sum_{i\ge0} F_1(i + 5, 0)x^i\\
&\quad + f_6\sum_{i\ge0} F_1(i + 6, 0)x^i + f_7\sum_{i\ge0} F_1(i + 7, 0)x^i\\
&\quad + f_9F_1(i + 9, 0)x^i.
\end{aligned}$$

By substituting $\sum_{i\ge0} F_1(i, 0)x^i$ by $H_1(x)$ and using the boundary conditions, we get

$$\begin{aligned}
&\frac{1}{x^{12}}\left(H_1(x) - \sum_{i=0}^{11} x^i\right)\\
&= f_0H_1(x) + \frac{f_1}{x}\left(H_1(x) - \sum_{i=0}^{0} x^i\right) + \frac{f_2}{x^2}\left(H_1(x) - \sum_{i=0}^{1} x^i\right)\\
&\quad + \frac{f_3}{x^3}\left(H_1(x) - \sum_{i=0}^{2} x^i\right) + \frac{f_4}{x^4}\left(H_1(x) - \sum_{i=0}^{3} x^i\right)\\
&\quad + \frac{f_5}{x^5}\left(H_1(x) - \sum_{i=0}^{4} x^i\right) + \frac{f_6}{x^6}\left(H_1(x) - \sum_{i=0}^{5} x^i\right)\\
&\quad + \frac{f_7}{x^7}\left(H_1(x) - \sum_{i=0}^{6} x^i\right) + \frac{f_9}{x^9}\left(H_1(x) - \sum_{i=0}^{8} x^i\right).
\end{aligned}$$

Therefore,

$$H_1(x) = \frac{\sum_{j=5}^{11}\sum_{i=12-j}^{7} f_i x^j + \sum_{i=3}^{11} f_9 x^i - \sum_{i=0}^{11} x^i}{\sum_{i=5}^{12} f_{12-i}x^i + f_9x^3 - 1}.$$

Now we compute $G_1(x, y) = \sum_{i,j\ge0} F_1(i, j)x^iy^j$. From the recurrence relation of $F_1(\alpha, \beta)$, we obtain

$$\begin{aligned}
&\sum_{i,j\ge0} F_1(i + 12, j + 1)x^iy^j\\
&= s\sum_{i,j\ge0} F_1(i, j)x^iy^j + f_0\sum_{i,j\ge0} F_1(i, j + 1)x^iy^j\\
&\quad + f_1\sum_{i,j\ge0} F_1(i + 1, j + 1)x^iy^j\\
&\quad + f_2\sum_{i,j\ge0} F_1(i + 2, j + 1)x^iy^j\\
&\quad + f_3\sum_{i,j\ge0} F_1(i + 3, j + 1)x^iy^j\\
&\quad + f_4\sum_{i,j\ge0} F_1(i + 4, j + 1)x^iy^j\\
&\quad + f_5\sum_{i,j\ge0} F_1(i + 5, j + 1)x^iy^j\\
&\quad + f_6\sum_{i,j\ge0} F_1(i + 6, j + 1)x^iy^j\\
&\quad + f_7\sum_{i,j\ge0} F_1(i + 7, j + 1)x^iy^j\\
&\quad + f_9\sum_{i,j\ge0} F_1(i + 9, j + 1)x^iy^j
\end{aligned}$$

By substituting $\sum_{i\ge0} F_1(i, 0)x^i$ and $\sum_{i,j\ge0} F_1(i, j)x^iy^j$ by $H_1(x)$ and $G_1(x, y)$, respectively, and using the boundary conditions, we get

$$\begin{aligned}
&\frac{1}{x^{12}y}\left(G_1(x, y) - H_1(x)\right)\\
&= sG_1(x, y) + \frac{f_0}{y}\left(G_1(x, y) - H_1(x)\right)
\end{aligned}$$

$$+\frac{f_1}{xy}(G_1(x,y)-H_1(x))+\frac{f_2}{x^2y}(G_1(x,y)-H_1(x))$$
$$+\frac{f_3}{x^3y}(G_1(x,y)-H_1(x))+\frac{f_4}{x^4y}(G_1(x,y)-H_1(x))$$
$$+\frac{f_5}{x^5y}(G_1(x,y)-H_1(x))+\frac{f_6}{x^6y}(G_1(x,y)-H_1(x))$$
$$+\frac{f_7}{x^7y}(G_1(x,y)-H_1(x))+\frac{f_9}{x^9y}(G_1(x,y)-H_1(x)).$$

Therefore,

$$G_1(x,y)=\frac{\sum_{i=5}^{12}f_{12-i}x^i+f_9x^3-1}{syx^{12}+\sum_{i=5}^{12}f_{12-i}x^i+f_9x^3-1}H_1(x)$$

This finishes the proof. □

Next we compute the generating function of the probability $F_2(\alpha,\beta)$ that Algorithm Parse_SPDM2 outputs β coefficients in $\mathbb{Z}_q$ for given α random bits.

Proposition 2: Let $F_2(\alpha,\beta)$ be the probability that Algorithm *Parse_SPDM2* outputs β coefficients in $\mathbb{Z}_q$ for given α random bits, where α and β are non-negative integers. We assume that the algorithm tries to output a coefficient only when more than or equal to twelve random bits are left. And let $G_2(x,y)$ be the generating function of $F_2(\alpha,\beta)$. Then

$$G_2(x,y)=\frac{f_0x^{12}+f_1x^8+f_2x^4-1}{sx^{12}y+f_0x^{12}+f_1x^8+f_2x^4-1}H_2(x),$$

where $s=3329/4096, f_0=15/4096, f_1=240/4096, f_2=512/4096$, and

$$H_2(x)=\frac{f_1\sum_{i=8}^{11}x^i+f_2\sum_{i=4}^{11}x^i-\sum_{i=0}^{11}x^i}{f_0x^{12}+f_1x^8+f_2x^4-1}.$$

Proof: The probability $F_2(\alpha,\beta)$ satisfies the following recurrence relation.

$$F_2(\alpha+12,\beta+1)$$
$$=sF_2(\alpha,\beta)+f_0F_2(\alpha,\beta+1)+f_1F_2(\alpha+4,\beta+1)$$
$$+f_2F_2(\alpha+8,\beta+1)$$

Here, $F_2(i,0)=1$ for $0\le i<12$ and $F_2(i,j)=0$ for $0\le i<12$ and $1\le j$. These are the boundary conditions.

Before computing the generating function of $F_2(\alpha,\beta)$, we first compute the generating function of $F_2(\alpha,0)$. The probability $F_2(\alpha,0)$ satisfies the following recurrence relation.

$$F_2(\alpha+12,0)$$
$$=f_0F_2(\alpha,0)+f_1F_2(\alpha+4,0)+f_2F_2(\alpha+8,0)$$

Let $H_2(x)=\sum_{i\ge0}F_2(i,0)x^i$ be the generating function of $F_2(\alpha,0)$. From the recurrence relation of $F_2(\alpha,0)$, we obtain

$$\sum_{i\ge0}F_2(i+12,0)x^i$$
$$=f_0\sum_{i\ge0}F_2(i,0)x^i+f_1\sum_{i\ge0}F_2(i+4,0)x^i$$
$$+f_2\sum_{i\ge0}F_2(i+8,0)x^i.$$

By substituting $\sum_{i\ge0}F_2(i,0)x^i$ by $H_2(x)$ and using the boundary conditions, we get

$$\frac{1}{x^{12}}\left(H_2(x)-\sum_{i=0}^{11}x^i\right)$$
$$=f_0H_2(x)+\frac{f_1}{x^4}\left(H_2(x)-\sum_{i=0}^{3}x^i\right)$$
$$+\frac{f_2}{x^8}\left(H_2(x)-\sum_{i=0}^{7}x^i\right).$$

Therefore,

$$H_2(x)=\frac{f_1\sum_{i=8}^{11}x^i+f_2\sum_{i=4}^{11}x^i-\sum_{i=0}^{11}x^i}{f_0x^{12}+f_1x^8+f_2x^4-1}.$$

Now we compute $G_2(x,y)=\sum_{i,j\ge0}F_2(i,j)x^iy^j$. From the recurrence relation of $F_2(\alpha,\beta)$, we obtain

$$\sum_{i,j\ge0}F_2(i+12,j+1)x^iy^j$$
$$=s\sum_{i,j\ge0}F_2(i,j)x^iy^j+f_0\sum_{i,j\ge0}F_2(i,j+1)x^iy^j$$
$$+f_1\sum_{i,j\ge0}F_2(i+4,j+1)x^iy^j$$
$$+f_2\sum_{i,j\ge0}F_2(i+8,j+1)x^iy^j.$$

By substituting $\sum_{i\ge0}F_2(i,0)x^i$ and $\sum_{i,j\ge0}F_2(i,j)x^iy^j$ by $H_2(x)$ and $G_2(x,y)$, respectively, and using the boundary conditions, we get

$$\frac{1}{x^{12}y}(G_2(x,y)-H_2(x))$$
$$=sG_2(x,y)+\frac{f_0}{y}(G_2(x,y)-H_2(x))$$
$$+\frac{f_1}{x^4y}(G_2(x,y)-H_2(x))+\frac{f_2}{x^8y}(G_2(x,y)-H_2(x)).$$

Therefore,

$$G_2(x,y)=\frac{f_0x^{12}+f_1x^8+f_2x^4-1}{sx^{12}y+f_0x^{12}+f_1x^8+f_2x^4-1}H_2(x).$$

This finishes the proof. □

Finally, we compute the generating function of the probability $F_3(\alpha,\beta)$ that Algorithm Parse_SPDM3 outputs β coefficients in $\mathbb{Z}_q$ for given α random bits.

Proposition 3: Let $F_3(\alpha,\beta)$ be the probability that Algorithm *Parse_SPDM3* outputs β coefficients in $\mathbb{Z}_q$ for given α random bits, where α and β are non-negative integers. We assume that the algorithm tries to output a coefficient only when more than or equal to twelve random bits are left. And let $G_3(x,y)$ be the generating function of $F_3(\alpha,\beta)$. Then

$$G_3(x,y)=\frac{cx^2-x^2+cx-x-1}{ax^3y+bx^3+cx-1}H_3(x),$$

where $a = 3329/4096$, $b = 255/4096$, $c = 512/4096$, and

$$H_3(x) = \frac{c\sum_{i=4}^{11} x^i - \sum_{i=0}^{11} x^i}{bx^{12} + cx^4 - 1}.$$

Proof: The probability $F_3(\alpha, \beta)$ satisfies the following recurrence relation.

$$F_3(\alpha + 12, \beta + 1) = aF_3(\alpha, \beta) + bF_3(\alpha, \beta + 1) + cF_3(\alpha + 8, \beta + 1)$$

Here, $F_3(i, 0) = 1$ for $0 \le i < 12$ and $F_3(i, j) = 0$ for $0 \le i < 12$ and $1 \le j$. These are the boundary conditions.

Before computing the generating function of $F_3(\alpha, \beta)$, we first compute the generating function of $F_3(\alpha, 0)$. The probability $F_3(\alpha, 0)$ satisfies the following recurrence relation.

$$F_3(\alpha + 12, 0) = bF_3(\alpha, 0) + cF_3(\alpha + 8, 0)$$

Let $H_3(x) = \sum_{i\ge 0} F_3(i, 0)x^i$ be the generating function of $F_3(\alpha, 0)$. From the recurrence relation of $F_3(\alpha, 0)$, we obtain

$$\sum_{i\ge 0} F_3(i + 12, 0)x^i = b\sum_{i\ge 0} F_3(i, 0)x^i + c\sum_{i\ge 0} F_3(i + 8, 0)x^i.$$

By substituting $\sum_{i\ge 0} F_3(i, 0)x^i$ by $H_3(x)$ and using the boundary conditions, we get

$$\frac{1}{x^{12}}\left(H_3(x) - \sum_{i=0}^{11} x^i\right) = bH_3(x) + \frac{c}{x^8}\left(H_3(x) - \sum_{i=0}^{7} x^i\right).$$

Therefore,

$$H_3(x) = \frac{c\sum_{i=4}^{11} x^i - \sum_{i=0}^{11} x^i}{bx^{12} + cx^4 - 1}.$$

Now we compute $G_3(x, y) = \sum_{i,j\ge 0} F_3(i, j)x^i y^j$. From the recurrence relation of $F_3(\alpha, \beta)$, we obtain

$$\sum_{i,j\ge 0} F_3(i + 12, j + 1)x^i y^j = a\sum_{i,j\ge 0} F_3(i, j)x^i y^j + b\sum_{i,j\ge 0} F_3(i, j + 1)x^i y^j + c\sum_{i,j\ge 0} F_3(i + 8, j + 1)x^i y^j.$$

By substituting $\sum_{i\ge 0} F_3(i, 0)x^i$ and $\sum_{i,j\ge 0} F_3(i, j)x^i y^j$ by $H_3(x)$ and $G_3(x, y)$, respectively, and using the boundary conditions, we get

$$\frac{1}{x^{12}y}\left(G_3(x, y) - H_3(x)\right) = aG_3(x, y) + \frac{b}{y}\left(G_3(x, y) - H_3(x)\right) + \frac{c}{x^8 y}\left(G_3(x, y) - H_3(x)\right).$$

Therefore,

$$G_3(x, y) = \frac{bx^{12} + cx^4 - 1}{ax^{12}y + bx^{12} + cx^4 - 1}H_3(x).$$

This finishes the proof. □

REFERENCES

[1] National Institute of Standards and Technology (2013), *Digital Signature Standard (DSS)*, Standard FIPS PUBS 184-6, U.S. Department of Commerce, Washington, DC, USA, Jul. 2013.
[2] D. Bleichenbacher, "On the generation of one-time keys in DL signature schemes," in *Proc. Work. Group Meeting*, 2000, p. 81.
[3] *Information Technology—Security Techniques—Random Bit Generation*, Standard ISO/IEC 18031, 2011.
[4] American National Standards Institute (2006), *Random Number Generation—Part 1: Overview and Basic Principles*, Standard ANSI X9.82-1-2006, R2013, Jul. 2006.
[5] National Institute of Standards and Technology (2015), *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, Standard 800-90A Revision 1, U.S. Department of Commerce, Washington, DC, USA, Jun. 2015.
[6] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber algorithm specifcations and supporting documentation," Third-round submission to the NIST's post-quantum cryptography standardization process," Nat. Inst. Standards Technol., Tech. Rep., 2020. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/Kyber-Round3.zip
[7] National Institute of Standards and Technology (2023), *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, Standard FIPS PUBS 202, U.S. Department of Commerce, Washington, DC, USA, Aug. 2015.
[8] B. Koo, D. Roh, and D. Kwon, "Converting random bits into random numbers," *J. Supercomput.*, vol. 70, no. 1, pp. 236–246, Oct. 2014.
[9] National Institute of Standards and Technology (2023), *Module-Lattice-Based Key-Encapsulation Mechanism Standard*, Standard FIPS PUBS 203, U.S. Department of Commerce, Washington, DC, USA, Aug. 2023.
[10] *Information Technology—Security Techniques—Encryption Algorithms—Part 2: Asymmetric Ciphers—Amendment 2*, Standard ISO/IEC 18033-2, 2006.

DONGYOUNG ROH (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in mathematics from the Korea Institute of Science and Technology, Daejeon, Republic of Korea, in 2011.

From 2011 to 2012, he was a Researcher with National Institute for Mathematical Sciences. Since 2012, he has been a Principal Researcher with The Affiliated Institute of Electronics and Telecommunications Research Institute. He is the author of more than ten articles, the editor of three ISO/IEC standards, and holds two patents. His research interests include designing and analyzing cryptographic algorithms and relations between discrete logarithm related problems.

Dr. Roh was a recipient of the Mid-Career Professional in Global Achievement Awards by $(ISC)^2$, in 2020.

SANGIM JUNG (Member, IEEE) received the B.S. degree in mathematics from Sookmyung Women's University, Seoul, Republic of Korea, in 2010, and the M.S. degree in information security from Korea University, Seoul, in 2012. Since 2011, she has been a Senior Researcher with The Affiliated Institute of Electronics and Telecommunications Research Institute. Her research interest includes designing and analyzing cryptographic protocols.