To reject or not reject - that is the question. The case of BIKE post quantum KEM

Nir Drucker
IBM Research, Haifa, Israel
drucker.nir@gmail.com

Shay Gueron
University of Haifa, Israel,
and
Amazon, USA.
shay.gueron@gmail.com

Dusan Kostic
Amazon, USA.
dkostic@protonmail.com

***Abstract*—NIST post-quantum cryptography standardization project just entered its final Round 4, where three KEMs are evaluated for standardization, as alternatives. BIKE is one of them. This paper deals with several considerations around building an isochronous and constant-time implementation of the errors-vector generation (EVG) that is used by BIKE. The starting point is the Round 3 BIKE (Ver. 4.2), where a recently published timing attack motivated some changes toward the Round 4 submission. The easiest mitigation simply redefines the EVG to be isochronous. This approach was readily available (already in June 2022) in [1]. It requires only minor changes in the Round 3 specification and reference code, with no changes to the KATs. However, BIKE chose a different, newly proposed EVG method (with new KATs). It was integrated into the definition and reference code of the first Round 4 submission (Ver. 5.0) but turned out to be erroneous. We alerted NIST and the BIKE team about the problems, and proposed solutions. This responsible disclosure allowed the BIKE team to revisit the design decision per one of our solutions, modify the specifications document and the associated proof and submit a *revised* Round 4 submission (Ver. 5.1). NIST gracefully accepted the fixed specification as the submission. In this paper, we explore the problems, review and compare some engineering aspects associated with different approaches, present more alternatives and conclude with our critique and recommendations.**

***Index Terms*—bit flipping key encapsulation, BIKE, key encapsulation mechanism, rejection sampling, sampling, post-quantum cryptography.**

I. Introduction

Bit Flipping Key Encapsulation (BIKE) is a key encapsulation mechanism (KEM) and one of the submissions to the post-quantum cryptography (PQC) standardization project led by the National Institute for Standards and Technology (NIST). At the end of the third round, the KEM Kyber [2] was selected for standardization and NIST started a fourth round where it considers BIKE [3], Classical McEliece [4], and Hamming Quasi-Cyclic (HQC) [5] as alternative KEM candidates for standardization.

During the third round of the process, Guo et al. [6] showed an attack on BIKE and HQC, where here, we focus on Bit Flipping Key Encapsulation (BIKE). The attack relied on the fact that BIKE (and HQC) defined a non-isochronous algorithm as part of their flow, and hence their implementation. Specifically, BIKE's encapsulation (Encaps) and decapsulation (Decaps) need to generate a uniform random *errors-vector*. This is a binary vector of length $2r$ that has a fixed and predetermined number t of nonzero bits ("weight"), for some parameters r and t. This is done by: a) generating a set $wlist$ of t (pseudo-)random distinct indices between 0 and $(2r-1)$; b) converting $wlist$ to the desired binary errors-vector by setting t bits of the (initially zero) errors-vector at the positions indicated by the elements of $wlist$.

The parameter r is an odd prime so $2r$ is not a power of 2. Thus, the generation of $wlist$ was executed by using a standard *rejection sampling* method (Alg. 1). Starting with $wlist = \phi$, draw a chunk of $B \geq \lceil \log_2(2r) \rceil$ (depending on the parameters and implementation convenience) bits uniformly at random as a candidate index i. This sample is rejected if $i > 2r-1$ or if $i \in wlist$. Otherwise, it is added to $wlist$. The process continues until $wlist$ has exactly t elements (indices). The number of draws until t distinct values are collected is a random variable. In other words, this rejection sampling algorithm is non-isochronous, because it does not run in a fixed number of steps.

The non-isochronous rejection sampling makes the decapsulation non-isochronous *even if* all the other steps of the decapsulation flow are carried out in constant-time (CT). The attack of Guo et al. [6] showed that the variable time decapsulation leaks information that can lead to a timing attack on BIKE (and more so, on HQC).

In response to the attack of [6], we introduced (in June 2022) a mitigation in our Additional and Optimized implementation package [1] of BIKE. This mitigation did not change the observed outputs of BIKE. Specifically, it did not change the

Algorithm 1 WSHAKE256-PRF$(seed, wt, len)$ [7][Alg. 3]

require: $seed$, wt (32 bits), len
ensure: A list $(wlist)$ of wt bit-positions in $[0, 1, \ldots, len-1]$
1: **procedure** WSHAKE256-PRF($seed$, wt, len)
2: $wlist = \phi$, $ctr = 0$, $i = 0$
3: $s = \text{SHAKE256-Stream}(seed; \infty)$ ▷ ∞ - "sufficiently large"
4: $mask = 2^{\lceil \log_2 len \rceil} - 1$
5: **while** $ctr < wt$ **do**
6: $pos = s[32(i+1) - 1 : 32i]$ & $mask$
7: **if** $(pos < len)$ and $(pos \notin wlist)$ **then**
8: $wlist = wlist \cup pos$
9: $ctr = ctr + 1$
10: $i = i + 1$
11: **return** $wlist$

known answers tests (KATs) of the round three specifications document.

In Section III of this paper, we present this mitigation in detail and discuss some trade-offs between code simplicity, design simplicity, and performance impact.

A different strategy to fix the non-isochronous design of BIKE [7] was proposed by Sendrier in [8]. It replaces the rejection sampling in (*only*) the encapsulation and the decapsulation with an algorithm that has no rejection. On the other hand, it generates a non-uniform distribution of the indices (shown in [8] to have a negligible effect on the overall security). Section IV below explains why we argue that the design of [8] is not a good engineering trade-off for a KEM.

The design of [8] was the chosen path for the fourth round BIKE submission. The first submission package (specifications document, reference implementation, and KATs) was submitted to NIST as the Round 4 BIKE [9][Oct. 4, 2022], but was erroneous. We brought this fact to the attention of NIST and the BIKE team and proposed alternatives to fix the problem. As a result, the submission was fixed in [3][Oct. 10, 2022], with one of our proposed options. The modified version uses a *biased* fixed-weight sampling for the errors-vector generation in Encaps and Decaps, and also a biased fixed-weight sampling in the key generation algorithm. This is described in Section V.

II. Preliminaries and Notation

Let $\mathbb{F}_2$ be the finite field of characteristic 2. Let $\mathcal{R}$ be the polynomial ring $\mathbb{F}_2[X]/\langle X^r-1\rangle$. Polynomials in $\mathcal{R}$ are viewed interchangeably as binary vectors. The Hamming weight of every element $v \in \mathcal{R}$ is denoted by $wt(v)$. We denote protocol failures by $\perp$. Uniform random sampling from a set W is denoted by $w \xleftarrow{\$} W$ and sampling from a distribution $\mathcal{D}$ over W is denoted by $w \xleftarrow{\mathcal{D}} W$.

Fixed-weight sampling. Let len, wt be positive integers with $len > wt$, and let $\mathcal{S}_{len,wt}$ denote the set of $\binom{len}{wt}$ subsets of $\{0,1,\ldots,len-1\}$ with cardinality wt. Let $\mathcal{D}$ be a probability distribution over $\mathcal{S}_{len,wt}$. An algorithm that samples an element from $\mathcal{S}_{len,wt}$ according to a distribution $\mathcal{D}$, i.e., executing $\xleftarrow{\mathcal{D}} \mathcal{S}_{len,wt}$, is called a fixed-weight sampling algorithm. The notation $\xleftarrow{\mathcal{D}} \mathcal{S}_{len,wt}(m)$ refers to pseudorandom fixed-weight sampling starting from a seed m, The notation $\xleftarrow{\$} \mathcal{S}_{len,wt}$ means the special case where $\mathcal{D}$ is the uniform distribution.

The KEM BIKE. The KEM BIKE [7][v4.2] is defined with parameters r, t, w, d ($d = w/2$) and is described in Fig. 1. The functions $\mathbf{K}$ and $\mathbf{L}$ are modeled as random oracles, with a concrete associated instantiation. The function Decode takes as input a syndrome $s \in \mathcal{R}$ and parity polynomials $(h_0, h_1) \in (\mathcal{S}_{r,d})^2$ and outputs a sparse vector $(e_0, e_1) \in \mathcal{S}_{2r,t}$.

BIKE uses fixed-weight sampling for KeyGen, Encaps and Decaps as follows.

- Step 2 of KeyGen invokes, twice, a sampling algorithm denoted `Sample-Gen` that implements $\xleftarrow{\$} \mathcal{S}_{r,d}$.
- Encaps and Decaps invoke a sampling algorithm denoted `Sample-ED` that implements $\xleftarrow{\$} \mathcal{S}_{2r,t}$.

$(sk, \sigma, h) \leftarrow \text{KeyGen}()$

1) $\sigma \xleftarrow{\$} \{0,1\}^{256}$
2) $h_0, h_1 \xleftarrow{\mathcal{D}_1} \mathcal{S}_{r,d}, \mathcal{S}_{r,d}$
3) $sk = (h0, h1, \sigma)$
4) $pk = h = h_1 h_0^{-1}$
5) Return (sk, pk)

$(C, K) \leftarrow \text{Encaps}(h)$

1) $m \xleftarrow{\$} \{0,1\}^{256}$
2) $e = (e_0, e_1) \xleftarrow{\mathcal{D}_2} \mathcal{S}_{n,t}(m)$
3) $C = (c_0, c_1) = (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$
4) $K = \mathbf{K}(m, C)$
5) Return (C, K)

$m = \text{Decaps}(sk, \sigma, h, C)$

1) m' = Decode(sk, C) // Or $\perp$ on decoding failure
2) $e' = (e'_0, e'_1) \xleftarrow{\mathcal{D}_2} \mathcal{S}_{n,t}(m')$
3) $C' = (c'_0, c'_1) = (e'_0 + e'_1 h, m' \oplus \mathbf{L}(e'_0, e'_1))$
4) If $m' = \perp$ or $C \neq C'$ then $m' = \sigma$
5) $K = \mathbf{K}(m', C)$
6) Return K

Fig. 1. A schematic description of the KEM BIKE. For the third round BIKE (v4.2) [7], $\mathcal{D}_1 = \mathcal{D}_2 = \$$. For the fourth round BIKE (v5.0) [9], $\mathcal{D}_1 = \$$ and $\mathcal{D}_2$ is some "biased" distribution. For the *fixed* fourth round BIKE (v5.1) [3], $\mathcal{D}_1$ and $\mathcal{D}_2$ are two (different) "biased" distributions. See the notation in Section II and the explanations in the text.

BIKE [7][v4.2] specifies a fixed weight algorithm (Alg. 3 therein) for `Sample-Gen` and `Sample-ED`. It is listed as Alg. 1 above. It uses the SHAKE256-Stream function to generate psuedorandom data from an input seed *seed*.

A. Isochronous and CT implementations

This paper describes several considerations that are relevant to an isochronous and CT implementation of BIKE. Here, the term "constant-time" implementation refers to code that mitigates potential leaks from the micro-architectural features of the processor (e.g., memory access patterns and branches). The term "isochronous" refers to the overall number of steps of an algorithm, which directly affects the overall execution time of its implementation.

For example, BIKE [7][Alg. 3] is not isochronous because it includes a "while" loop that makes the number of steps unspecified apriori, and in reality, with a dependence on the seed (technicality it does not specify any bound on the number of steps, but this is not critical to the current discussion). Apparently, isochronous errors-vector generation (EVG) was not perceived as a required property of BIKE. Specifically, BIKE specifications document explicitly recommends using BIKE with ephemeral keys [7][e.g., on pages 10, 11, 12, 34], where isochronous EVG is assumed to be unnecessary. Furthermore,

alternatives to the non-isochronous EVG algorithm were also proposed e.g., in [10].

The attack of [6]. Qian Guo et al. [6] described an attack on BIKE and HQC. This attack leverages the non-isochronous property of [7][Alg. 3] when using BIKE with a fixed key-pair, despite BIKE's recommendation (see above). Since BIKE's reference code [7] is not a CT implementation, timing attacks, or any other attack on it, are irrelevant. However, they are relevant to our Additional and Optimized implementations package [1], which is referenced in [7].

Unified Sampling. Since there may be multiple ways to define (and execute) a uniform-random fixed-weight sampling algorithm, `Sample-ED` and `Sample-Gen` do not have to be identical. However, for design simplicity and minimalism, BIKE [7] defined them to be the same algorithm, taking different parameters.

Isochronous algorithms and isochronous and CT implementations. A secure BIKE implementation must execute `Sample-ED` and `Sample-Gen` in CT. However, a CT implementation does not necessarily need to be isochronous i.e., always execute a fixed number of steps. It is possible that the number of steps does not leak information that is relevant to an attacker. This is the case with the definition of `Sample-ED` and `Sample-Gen`, and hence their accompanying CT implementation was non-isochronous.

Specifically, for Alg. 1, the number of samples needed to successfully finish this algorithm is an unbounded random variable, i.e., the algorithm may (theoretically) never finish. Therefore, we need to set the limit on the number of samples the algorithm will perform in case it keeps failing to produce enough valid indices. Let the maximum number of samples be X, and assume it is pre-defined in the context of the algorithms hereafter[1]. Then we have two ways to implement the algorithm:

- **Maximum sampling number (MSN).** Generate samples until we have t distinct indices, but stop after X samples despite the outcome. Output an error indicator if the number of distinct values in the list is less than t even after X selections. Here, the number of steps is not fixed (i.e., the algorithm is not isochronous), but the algorithm terminates in a predetermined time frame.
- **Fixed sampling number (FSN).** Generate exactly X samples. Output an error indicator if the number of distinct values in the list is less than t. This makes the algorithm flow isochronous (assuming that the elements of the implementation run in CT). The choice of X controls the success probability, i.e., the probability of finding at least t distinct values in exactly X steps. Increasing X increases the success probability but, at the same time, increases the run-time of the algorithm. This run-time is (also) affected by how quickly we can generate random values in Step 1a.

[1]In [1], the constant X is named `MAX_RAND_INDICES_T`. We use the notation X for brevity.

Implementation complications. A secure CT and isochronous implementation needs to face two challenges: hide the timing and memory access in both Step 1 and Step 2. These challenges need a carefully-written code to avoid information leaks. If both steps are done in CT, then the FSN version hides the overall-run-time of the algorithm, while the MSN version does not. The implementation difference between the MSN and the FSN variants is minor and it is relatively easy to accommodate both.

III. Mitigating the timing attack of [6].

The obvious mitigation to the attack [6] is to use the FSN version of the EVG, with some predetermined value of X. This value does not change the *required* uniform distribution property of the generated errors-vector, for all cases where enough valid and distinct samples are collected in X samples. This is described in Alg. 2.

Algorithm 2 NEW_WSHAKE256-PRF$(seed, wt, len, X)$

Require: $seed$, wt, len, X
Ensure: A list $(wlist)$ of wt bit-positions in $[0, 1, \ldots, len-1]$
1: **procedure** NEW_WSHAKE256-PRF($seed$, wt, len, X)
2: $wlist = \phi$; $ctr = 0$
3: $s_0, s_1, \ldots, s_{X-1} =$ SHAKE256-Stream$(seed; 16X)$
4: $mask = 2^{\lceil \log_2 len \rceil} - 1$
5: **for** $i = 0 \ldots X-1$ **do**
6: $pos = si \ \& \ mask$
7: **if** $(pos < len)$ and $(pos \notin wlist)$ and $(ctr < wt)$ **then**
8: $wlist = wlist \cup pos$
9: $ctr = ctr + 1$
10: **return** $ctr == wt \ ? \ wlist \ : \ \text{error}$

Selecting a "good" value for X. Selecting X boils down to the following combinatorial exercise.

> Given parameters len, X, wt, $B \geq (\lceil \log_2 len \rceil - 1)$, compute the success probability of Alg. 2 (i.e., the probability that line 10 returns the list $wlist$).

The computations are given in appendix A. To illustrate, consider BIKE Level 1 where in Encaps and Decaps we have $len = 2r = 24{,}646$, $B = 15$ and $wt = 134$. Having these three parameters fixed, in Table I we compute the success probability $p(X)$ for different values of X. For $X = 327$, $p(X) \geq 1 - 2^{-128}$ and for $X = 271$, $p(X) \geq 1 - 2^{-64}$.

Observations.

1) A secure implementation of BIKE can still use Alg. 1 for the encapsulation because the number of steps of the sampling does not depend on the secret key. The key generation phase (which is executed only once) can also use Alg. 1.
2) The choice of X for Alg. 2 in decapsulation is an *engineering consideration* and not a security consideration. Indeed, when X samples do not suffice for collecting at least t distinct indices, the observed behavior would be exactly the same (oblivious) behavior as with a decoding failure (with no timing difference). Thus, we can choose X to be the smallest value that leads to a tolerable failure rate from the engineering viewpoint. For

example, failure with probability 2^{-64} seems more than adequate.

The (relative) overhead of isochronous sampling. Table I also shows the performance overhead of Alg. 2 compared to the non-isochronous version of Alg. 1, measured in number of processor cycles (on and Intel(R) Xeon(R) Platinum 8175M CPU @ 2.50GHz processor). To put the numbers in perspective, the performance overhead introduced in the decapsulation functionality ranges from 0.05 to 0.3% for Level 1, and from 0.03 to 0.2% for Level 3 parameter sets of BIKE (depending on the choice of X). Furthermore, even if we choose to use Alg. 2 in encapsulation as well, as we do in [1] for code simplicity, the overhead (in the encapsulation) is in the range of $0.7 - 4.1\%$ and $0.6 - 4.9\%$ for Level 1 and 3 respectively (depending on the choice of X).

IV. The sampling method of [8]

A different method for fixed-weight sampling was proposed by N. Sendrier in [8][Alg. 5] and prescribed to be used in BIKE during Encaps and Decaps and we describe it below as Alg. 3. This algorithm samples from a non-uniform ("biased") distribution that is defined through the algorithm itself. It is shown in [8] that the impact of the biased sampling (compared to uniform sampling) on BIKE's security is negligible.

Algorithm 3 FY$(seed, len, wt)$

Require: $seed$, wt, len
Ensure: A list $(wlist)$ of wt bit-positions in $[0, 1, \ldots, len - 1]$
1: **procedure** NEW_WSHAKE256-PRF($seed$, wt, len, X)
2: $\quad wlist = \phi$
3: $\quad s_0, \ldots, s_{wt-1} = \text{SHAKE256-Stream}(seed, 32 \cdot wt)$
4: $\quad$ **for** $i = (wt - 1), \ldots, 0$ **do**
5: $\quad\quad pos = i + \lfloor (len - i) s_i / 2^{32} \rfloor$
6: $\quad\quad wlist = wlist \cup (pos \in wlist)\ ?\ i\ :\ pos$
7: $\quad$ **return** $wlist$

The advantage of this sampling method is that it is inherently isochronous because there is no rejection of samples and hence, the number of obtained and processed samples is a predetermined value (specifically, for Encaps and Decaps it is t). The method was claimed to be superior to the rejection sampling method because it would prevent implementors to make the mistake of not enforcing a fixed number of steps in the implementation. It was also conjectured to be faster than the rejection sampling method.

Note that the sampling is only a part of the errors-vector generation. Alg. 3 has certain benefits, but it doesn't address the remaining challenge of converting the list of indices to an errors-vector in *CT*. Moreover, performance-wise, this second step dominates the runtime of the errors-vector generation.

Let us review this proposal from an engineering point of view. The proposal [8] does not specify any concrete algorithm for sampling that is required in key generation. It is therefore reasonable to assume that the intention was to keep using the rejection sampling algorithm specified in BIKE v4.2 [7] (given in Alg. 1 above). Regardless, any algorithm that executes $\xleftarrow{\$}$, uniform sampling, is by definition *different* from any algorithm that executes non-uniform sampling, in particular, sampling from the biased distribution defined in [8] (Alg. 3 above). We doubt that this is a wise design choice for a KEM that wishes to be standardized and then expected to be used ubiquitously, for a very long period of time. The reason is the following implication of the design:

> Every implementation of BIKE (software, hardware, firmware) would necessarily be burdened by the requirement to have to develop, maintain and implement two different sampling routines (at whatever cost - hardware included).

This choice certainly conflicts with the minimalism principle that we believe is a component of design best-practices, which we try to follow in our designs. Moreover, minimalism was also the design choice of BIKE where for [7][v4.2] BIKE team "narrowed down the set of BIKE variants to a single variant" and stated that "the pseudorandom generation uses a SHAKE256 implementation [..] Therefore, BIKE only relies on one single symmetric cryptographic primitive [..] This design choice is especially beneficial for hardware devices to reduce the overall footprint by instantiating only one symmetric primitive."

V. BIKE v5.0 spec [9] problem and the responsible disclosure

BIKE team released (to NIST) the fourth round package [9][v5.0] (Oct. 4, 2022). The specifications document refers to [8] as the source for the modifications. Unfortunately, the submission was erroneous, as we explain in the following paragraphs. The problem was revealed while we were trying to update our Additional constant-time Implementation of [1] to match the new specifications document (v5.0), where we discovered that the reference code and hence the KATs were erroneous (assuming that the specifications document is the definition of BIKE). The reason was that the reference implementation applied Alg. 3 to Encaps and Decaps (as described in the specifications document) *but also* to the key generation method. We disclosed this problem to NIST and the BIKE team (Oct. 7, 2022), explained the mistake, and proposed three options to resolve the issue:

- Option 1. Change the reference implementation to match the definition of [9] (i.e., to execute the proposal of [8]). This requires the code to have two sampling routines (uniform sampling and biased sampling of Alg. 3), and a change in the KATs.
- Option 2. Change the definition of BIKE to use biased sampling for Encaps, Decaps, and KeyGen. In this case, the reference implementation would automatically match the (revised) specifications document (with no change in the KATs).
- Option 3. Revert to the third round BIKE [7] design, and use Alg. 2 for Encaps, Decaps, and KeyGen.

We added the following remarks. As pointed out above, the first option (by [8]) would burden every current and future implementation of BIKE (software, hardware, firmware)

TABLE I
CAPTION

Fail probability	X for L1	Overhead (cycles)	X for L3	Overhead (cycles)
2^{-192}	N/A	N/A	488	10617
2^{-128}	327	3943	N/A	N/A
2^{-96}	300	2621	405	3115
2^{-80}	286	2156	389	2277
2^{-64}	271	1397	373	1460
2^{-48}	255	639	354	1408

with having to develop, maintain and implement two different sampling routines. The second option is also problematic: the potentially revised version of the spec would have no security proof because [8] refers to a different design. Thus, a different proof would be needed, and it must account for the cumulative effect of the biased sampling for Encaps, Decaps, and for KeyGen.

We also commented the following: even if such a proof is eventually provided, a biased-sampling-everywhere modified version of BIKE will end up as a cryptographic design that intentionally generates keys from a non-uniform distribution. This seems like an awkward design statement, certainly with the given motivation.

We suggested that Option 3 was, in our opinion, the best way to handle the situation (a posteriori, and to begin with).

BIKE team chose to adopt Option 2, released the amended specifications document [3] on Oct. 10, 2022, and submitted it to NIST.

VI. Summary

This paper discusses several aspects regarding the modification of the sampling algorithms in the latest version of BIKE [3], [9]. Specifically, we argue that it was enough to tweak the original rejection sampling method for mitigating the attack [6] while achieving better design in terms of engineering considerations.

After the announcement of [3], we made the necessary updates in our Additional and Optimized package [1] and were able to measure the performance impact. As expected the run-time of the biased sampling method and the isochronous rejection sampling is similar and in any case has a very small effect on the overall performance of BIKE methods.

It is interesting to note that some problems still exist with BIKE [3], even after that necessary amendments were introduced. The specifications document refers to [8] as the source for the biased sampling method. The justification for the security impact of the new sampling is split over the specifications document where the key generation part is described, and [8] where the encapsulation is handled. However, the algorithm defined in the specifications document (Alg. 3 therein) is different from Alg. 5 in [8]. The difference stems from an idea of [11] where a modular reduction is replaced by a multiplication and a shift. It is not clear if the distributions obtained by these two algorithms are the same. Therefore, the security proof of BIKE does not necessarily hold and additional clarifications need to be made.

VII. Acknowledgments

This research was partly supported by: NSF-BSF Grant 2018640; The Israel Science Foundation (grant No. 3380/19); The Center for Cyber Law and Policy at the University of Haifa, in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

References

[1] Nir Drucker, Shay Gueron, and Kostic Dusan. Additional implementation of BIKE (Bit Flipping Key Encapsulation), commit 40519b8338ebe1f7bcd0efd8419a180642d94aa4. https://github.com/awslabs/bike-kem, 2022.

[2] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. BIKE: Bit Flipping Key Encapsulation v5.1. 2021. https://pq-crystals.org/kyber/index.shtml.

[3] Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. BIKE: Bit Flipping Key Encapsulation v5.1. oct 2022. https://bikesuite.org/files/v5.0/BIKE_Spec.2022.10.10.1.pdf.

[4] Martin R. Albrecht, Daniel J. Bernstein, Chou Tung, Cid Carlos, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. BIKE: Bit Flipping Key Encapsulation v5.1. 2020. https://classic.mceliece.org/nist.html.

[5] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Bidoux Lo ic, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardoand Persichetti, and Gilles Zémor. Hamming Quasi-Cyclic (HQC). 2017. https://pqc-hqc.org/doc/hqc-specification_2017-11-30.pdf.

[6] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):223–263, Jun. 2022. https://doi.org/10.46586/tches.v2022.i3.223-263.

[7] Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. BIKE: Bit Flipping Key Encapsulation v4.2a. Oct 2021. https://bikesuite.org/files/v4.2/BIKE_Spec.2021.09.29.1.pdf.

[8] Nicolas Sendrier. Secure sampling of constant-weight words – application to bike. Cryptology ePrint Archive, Paper 2021/1631, 2021. https://eprint.iacr.org/archive/2021/1631/20220927:074053.

[9] Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. BIKE: Bit Flipping Key Encapsulation v5. oct 2022. https://bikesuite.org/files/v5.0/BIKE_Spec.2022.10.04.1.pdf.

[10] Nir Drucker and Shay Gueron. Generating a Random String with a Fixed Weight. In Shlomi Dolev, Danny Hendler, Sachin Lodha, and Moti Yung, editors, *Cyber Security Cryptography and Machine Learning*, pages 141–155, Cham, 2019. Springer International Publishing. https://doi.org/10.1007/978-3-030-20951-3_13.

[11] Daniel Lemire. Fast random integer generation in an interval. *ACM Trans. Model. Comput. Simul.*, 29(1), jan 2019. https://doi.org/10.1145/3230636.

APPENDIX A
REJECTION SAMPLING SUCCESS PROBABILITY

The problem.

Given parameters N, B, X, and t compute the success probability of a Alg. 2 (i.e., the probability that line 10 returns the list $wlist$).

The computations are detailed as follows. Let p_{valid} denote the probability that a B-bit uniform random sample is a valid sample, i.e., the sample is in the range $[0, N-1]$. Then

$$p_{valid} = \frac{N}{2^B}. \tag{1}$$

Let p_1 denote the probability that exactly k out of n random B-bit samples are valid. Then,

$$p_1(k,n) = \binom{n}{k} \cdot p_{valid}^k \cdot (1-p_{valid})^{n-k}. \tag{2}$$

Let p_2 denote the probability that among k *valid* samples there are *exactly* d distinct values ($d \leq k$). Then,

$$p_2(d,k) = \binom{N}{d} \sum_{i=0}^{d} (-1)^i \binom{d}{i} \left(\frac{d-i}{N}\right)^k. \tag{3}$$

Let p_3 denote the probability that among k *valid* samples there are *at least* d distinct values ($d \leq k$). Then,

$$p_3(d,k) = \sum_{i=d}^{k} p_2(i,k). \tag{4}$$

Let p_4 denote the probability that among n random B-bit samples there are at least d valid and distinct values. Then,

$$p_4(d,n) = \sum_{i=d}^{n} p_3(d,i) \cdot p_1(i,n). \tag{5}$$

Finally, the probability to collect get less than t values in X random B-bit samples is:

$$p(X) = 1 - p_4(t,X). \tag{6}$$