On Constant-Time QC-MDPC Decoders with Negligible Failure Rate

Nir Drucker[1,2](✉), Shay Gueron[1,2], and Dusan Kostic[3]

[1] University of Haifa, Haifa, Israel
drucker.nir@gmail.com
[2] Amazon, Seattle, USA
[3] EPFL, Lausanne, Switzerland

Abstract. The QC-MDPC code-based KEM Bit Flipping Key Encapsulation (BIKE) is one of the Round-2 candidates of the NIST PQC standardization project. It has a variant that is proved to be IND-CCA secure. The proof models the KEM with some black-box ("ideal") primitives. Specifically, the decapsulation invokes an ideal primitive called "decoder", required to deliver its output with a negligible Decoding Failure Rate (DFR). The concrete instantiation of BIKE substitutes this ideal primitive with a new decoding algorithm called "Backflip", that is shown to have the required negligible DFR. However, it runs in a variable number of steps and this number depends on the input and on the key. This paper proposes a decoder that has a negligible DFR and also runs in a fixed (and small) number of steps. We propose that the instantiation of BIKE uses this decoder with our recommended parameters. We study the decoder's DFR as a function of the scheme's parameters to obtain a favorable balance between the communication bandwidth and the number of steps that the decoder runs. In addition, we build a constant-time software implementation of the proposed instantiation, and show that its performance characteristics are quite close to the IND-CPA variant. Finally, we discuss a subtle gap that needs to be resolved for every IND-CCA secure KEM (BIKE included) where the decapsulation has nonzero failure probability: the difference between average DFR and "worst-case" failure probability per key and ciphertext.

Keywords: BIKE · QC-MDPC codes · IND-CCA · Constant-time algorithm · Constant-time implementation

1 Introduction

BIKE [3] is a code-based Key Encapsulation Mechanism (KEM) using Quasi-Cyclic Moderate-Density Parity-Check (QC-MDPC) codes. It is one of the Round-2 candidates of the NIST PQC Standardization Project [16]. BIKE submission includes three variants (BIKE-1, BIKE-2, and BIKE-3) with three security levels for each one. Hereafter, we focus mainly on BIKE-1, at its Category 1 (as defined by NIST) security level.

M. Baldi et al. (Eds.): CBCrypto 2020, LNCS 12087, pp. 50–79, 2020.
https://doi.org/10.1007/978-3-030-54074-6_4

The decapsulation algorithm of BIKE invokes an algorithm that is called a decoder. The decoder is an algorithm that, given prescribed inputs, outputs an error vector that can be used to extract a message. There are various decoding algorithms and different choices yield different efficiency and DFR properties.

QC-MDPC Decoding Algorithms. We briefly describe the evolution of several QC-MDPC decoding algorithms. All of them are derived from the Bit-Flipping algorithm that is commonly attributed [10]. The Round-1 submission of BIKE describes the "One-Round" decoder. This decoder is indeed implemented in the accompanying reference code [3]. The designers of the constant-time Additional implementation [7] of BIKE Round-1 chose to use a different decoder named "Black-Gray"[1], with rationale as explained in [6]. The study presented in [20] explores two additional variants of the Bit-Flipping decoder: a) a parallel algorithm similar to that of [10], which first calculates some thresholds for flipping bits, and then flips the bits in all of the relevant positions, in parallel. We call this decoder the "Simple-Parallel" decoder; b) a "Step-by-Step" decoder (an enhancement of the "in-place" decoder described in [8]). It recalculates the threshold every time that a bit is flipped.

The Round-2 submission of BIKE uses the One-Round decoder (of Round-1) and a new variant of the Simple-Parallel decoder. The latter introduces a new trial-and-error technique called Time-To-Live (TTL). It is positioned as a derivative of the decoders in [20]. All of these decoders have some nonzero probability to fail in decoding a valid input. The average of the failure probability over all the possible inputs (keys and messages) is called Decoding Failure Rate (DFR). The KEMs of Round-1 BIKE were designed to offer IND-CPA security and to be used only with ephemeral keys. They had an approximate DFR of 10^{-7}, which is apparently tolerable in real systems. As a result, they enjoyed acceptable bandwidth and performance. Round-2 submission presented new variants of BIKE KEMs that provide IND-CCA security. Such KEMs can be used with static keys. The IND-CCA BIKE is based on three changes over the IND-CPA version: a) a transformation (called $FO^{\not\perp}$) applied to the key generation, encapsulation and decapsulation of the original IND-CPA flows (see [3][Section 6.2.1]); b) adjusted parameters sizes; c) invoking the Backflip decoder in the decapsulation algorithm.

Our Contribution

- We define Backflip$^+$ decoder as the variant of Backflip that operates with a fixed X_{BF} number of iterations (for some X_{BF}). We also define the Black-Gray decoder that runs with a given number of iterations X_{BG} (for some X_{BG}). Subsequently, we analyze the DFR of these decoders as a function of X_{BF} and X_{BG} and the block size (which determines the communication bandwidth of the KEM). The analysis finds a new set of parameters where Backflip$^+$ with $X_{BF} = 8, 9, 10, 11, 12$ and the Black-Gray decoder with

[1] This decoder appears in the pre-Round-1 submission "CAKE" (the BIKE-1 ancestor). It is due to N. Sendrier and R. Misoczki. The decoder was adapted to use the improved thresholds published in [3].

$X_{BG} = 3, 4, 5$ have an estimated average DFR of 2^{-128}. This offers multiple IND-CCA proper BIKE instantiation options.

- We build an optimized constant-time implementation of the new BIKE CCA flows together with a constant-time implementation of the decoders. This facilitates a performance comparison between the Backflip^{+} and the Black-Gray decoders. All of our performance numbers are based *only* on constant-time implementations. The comparison leads to interesting results. The Backflip^{+} decoder has a better DFR than the Black-Gray decoder if both of them are allowed to have a very large (practically unlimited) X_{BF} and X_{BG} values. These values do not lead to practical performance. However, for small X_{BF} and X_{BG} values that make the performance practical and DFR acceptable, the Black-Gray decoder is faster (and therefore preferable).
- The BIKE CCA flows require higher bandwidth and more computations compared to the original CPA flows, but the differences as measured on x86-64 architectures are not very significant. Table 1 summarizes the trade-off between the BIKE-1 block size (r), the estimated DFR and the performance of BIKE-1 decapsulation (with IND-CCA flows) using the Black-Gray decoder. It provides several instantiations/implementations choices. For example, with $X_{BG} = 4$ iterations and targeting a DFR of 2^{-64} (with $r = 11,069$ bits) the decapsulation with the Black-Gray decoder consumes 4.81M cycles. With a slightly higher $r = 11,261$ the decoder can be set to have only $X_{BG} = 3$ iterations and the decapsulation consumes 3.76M cycles.

Table 1. The BIKE-1 Level-1 block size r (in bits) for which the Black-Gray decoder achieves a target DFR with a specified number of iterations, and the decapsulation performance (in cycles; the precise details of the platform are provided in Sect. 5). A DFR of 2^{-128} is required for the IND-CCA KEM. The IND-CPA used with ephemeral keys can settle with higher DFR.

DFR		3 iterations	4 iterations	5 iterations
$2^{-23} \approx 10^{-7}$	r	$10,259$	$10,163$	$10,141$
	Cycles	3.50M	4.52M	5.53M
$2^{-30} \approx 10^{-9}$	r	$10,427$	$10,331$	$10,301$
	Cycles	3.52M	4.56M	5.63M
$2^{-40} \approx 10^{-12}$	r	$10,667$	$10,589$	$10,501$
	Cycles	3.55M	4.63M	5.69M
2^{-64}	r	$11,261$	$11,069$	$11,003$
	Cycles	3.76M	4.81M	5.96M
2^{-128}	r	$12,781$	$12,437$	$12,373$
	Cycles	4.06M	5.22M	6.47M

- The $FO^{\not\perp}$ transformation from QC-MDPC McEliece Public Key Encryption (PKE) to BIKE-1 IND-CCA relies on the assumption that the underlying

PKE is δ-correct [12] with $\delta = 2^{-128}$. The relation between this assumption and the (average) DFR that used in [3] is not yet addressed. We identify this gap and illustrate some of the remaining challenges.

The paper is organized as follows. Section 2 offers background, notation and surveys some QC-MDPC decoders. In Sect. 3 we define and clarify subtle differences between schemes using idealized primitives and concrete instantiations of the schemes. In Sect. 4 we explain the method used for estimating the DFR. We explain the challenges and the techniques that we used for building a constant-time implementation of IND-CCA BIKE in Appendix B. Section 5 reports our results for the DFR and block size study, and also the performance measurements of the constant-time implementations. The gap between the estimated DFR and the δ-correctness needed for IND-CCA BIKE is discussed in Sect. 6. Section 7 concludes this paper with several concrete proposals and open questions.

2 Preliminaries and Notation

Let $\mathbb{F}_2$ be the finite field of characteristic 2. Let $\mathcal{R}$ be the polynomial ring $\mathbb{F}_2[X]/\langle X^r - 1\rangle$. For every element $v \in \mathcal{R}$ its Hamming weight is denoted by $wt(v)$. The length of a vector w is denoted by $|w|$. Polynomials in $\mathcal{R}$ are viewed interchangeably also as square circulant matrices in $\mathbb{F}_2^{r\times r}$. For a matrix $H \in \mathbb{F}_2^{r\times r}$ let h_j denote its j-th column written as a row vector. We denote null values and protocol failures by $\perp$. Uniform random sampling from a set W is denoted by $w \xleftarrow{\$} W$. For an algorithm A, we denote its output by $out = A()$ if A is deterministic, and by $out \leftarrow A()$ otherwise. Hereafter, we use the notation $x.y\text{e}{-}z$ to denote the number $(x + \frac{y}{10}) \cdot 10^{-z}$.

2.1 BIKE-1

The computations of BIKE-1-(CPA/CCA) are executed over $\mathcal{R}$, where r is a given parameter. Let w and t be the weights of (h_0, h_1) in the secret key $h = (h_0, h_1, \sigma_0, \sigma_1)$ and the errors vector $e = (e_0, e_1)$, respectively. Denote the public key, ciphertext, and shared secret by $f = (f_0, f_1)$, $c = (c_0, c_1)$, and k, respectively. As in [3], we use $\mathbf{H}$, $\mathbf{K}$ to denote hash functions. Currently, the parameters of BIKE-1-CPA for NIST Level-1 are $r = 10,163$, $|f| = |c| = 20,326$ and for BIKE-1-CCA are $r = 11,779$, $|f| = |c| = 23,558$. In both cases, $|k| = 256$, $w = 142$, $d = w/2 = 71$ and $t = 134$. Figure 1 shows the BIKE-1-CPA and BIKE-1-CCA flows [3], see details therein.

2.2 The IND-CCA Transformation

Round-2 BIKE submission [3] uses the $FO^{\not\perp}$ conversion ([12] which relies on [9]) to convert the QC-MDPC McEliece PKE into an IND-CCA KEM BIKE-1-CCA. The submission claims that the proof results from [12][Theorems 3.1 and 3.4[2]].

[2] Theorems 3.1 and 3.4 appear only in the ePrint version [13] of [12]. In [12] they appear as Theorems 1 and 4, respectively.

<table>
<tr><th></th><th>BIKE-1 IND-CPA</th><th>BIKE-1 IND-CCA</th></tr>
<tr><td rowspan="3">Key generation</td><td colspan="2">$h_0, h_1 \xleftarrow{\$} \mathcal{R}$ of odd weight $wt(h_0) = wt(h_1) = w/2$</td></tr>
<tr><td>-</td><td>$\sigma_0, \sigma_1 \xleftarrow{\$} \mathcal{R}$</td></tr>
<tr><td colspan="2">$g \xleftarrow{\$} \mathcal{R}$ of odd weight (so $wt(g) \approx r/2$)
$(f_0, f_1) = (gh_1, gh_0)$</td></tr>
<tr><td rowspan="5">Encapsulation</td><td colspan="2">$m \xleftarrow{\$} \mathcal{R}$</td></tr>
<tr><td>$e_0, e_1 \xleftarrow{\$} \mathcal{R}$</td><td>$(e_0, e_1) = \mathbf{H}(mf_0, mf_1)$</td></tr>
<tr><td colspan="2">where $wt(e_0) + wt(e_1) = t$</td></tr>
<tr><td colspan="2">$(c_0, c_1) = (mf_0 + e_0, mf_1 + e_1)$</td></tr>
<tr><td>$k = \mathbf{K}(e_0, e_1)$</td><td>$k = \mathbf{K}(mf_0, mf_1, c_0, c_1)$</td></tr>
<tr><td rowspan="4">Decapsulation</td><td colspan="2">Compute the syndrome $s = c_0 h_0 + c_1 h_1$
$(e_o', e_1') \leftarrow$ decode(s, h_0, h_1)
If $wt\left((e_0', e_1')\right) \neq t$ or decoding failed then</td></tr>
<tr><td>return $\perp$</td><td>$k = \mathbf{K}(\sigma_0, \sigma_1, c)$</td></tr>
<tr><td colspan="2">else</td></tr>
<tr><td>$k = \mathbf{K}(e_0', e_1')$</td><td>$k = \mathbf{K}(c_0 + e_0', c_1 + e_1', c_0, c_1)$</td></tr>
</table>

Fig. 1. BIKE-1 IND-CPA/IND-CCA flows (full details are given in [3]).

These theorems use the term δ-correct PKEs. For a finite message space M, a PKE is called δ-correct when[3]

$$\mathbb{E}\left[\max_{m \in M} Pr\left[Decrypt(sk, c) \neq m \mid c \leftarrow Encrypt(pk, m)\right]\right] \leq \delta \qquad (1)$$

a KEMs is δ-correct if

$$Pr\left[Decaps(sk, c) \neq K | (sk, pk) \leftarrow Gen(), (c, K) \leftarrow Encaps(pk)\right] \leq \delta \qquad (2)$$

2.3 QC-MDPC Decoders

The QC-MDPC decoders discussed in this paper are variants of the Bit Flipping decoder [10] presented in Algorithm 1. They receive a parity check matrix $H \in \mathbb{F}_2^{r \times n}$ and a vector $c = mf + e$ as input[4]. Here, $c, mf, e \in \mathbb{F}_2^n$, mf is a codeword (thus, $H(mf)^T = 0$) and e is an error vector with small weight. The algorithm calculates the syndrome $s = eH^T$ and subsequently extracts e' from s. The goal of the Bit Flipping algorithm is to have e' such that $e' = e$.

Algorithm 1 consists of four steps: I) calculate some static/dynamic threshold (th) based on the syndrome (s) and the error (e) weights; II) compute the number of unsatisfied parity check equations (upc_i) for a given column $i \in \{0, \ldots, n-1\}$; III) Flip the error bits in the positions where there are more unsatisfied parity-check equations than the calculated threshold; IV) Recompute the syndrome.

[3] In BIKE-1, the secret key (sk) and public key (pk) are h and f, respectively.

[4] In BIKE-1, $n = 2r$, the parity-check matrix H is formed by the two circulant blocks (h_0, h_1), the vectors c, e, and f are defined as $c = (c_0, c_1)$, $e = (e_0, e_1)$, and $mf = (m \cdot f_0, m \cdot f_1)$.

We refer to Algorithm 1 as the Simple-Parallel decoder. The Step-By-Step decoder inserts Steps 4, 9 into the "`for`" loop (Step 5), i. e., it recalculate the threshold and the syndrome for every bit. The One-Round decoder starts with one iteration of the Simple-Parallel decoder, and then switches to the Step-by-Step decoder mode of operation.

Algorithm 1. e=BitFlipping(c, H)

```
    Input: Parity-check matrix $H \in \mathbb{F}_2^{r \times n}$, $c \in \mathbb{F}_2^n$, maxIter (maximal # of iterations),
    $u$ maximal syndrome weight
    Output: The error $e \in \mathbb{F}_2^n$
    Exception: "decoding failure" return $\perp$
 1: procedure BitFlipping($c$, $H$)
 2:     s = $Hc^T$, e = 0, itr = 0
 3:     while ($wt(s) > u$) and (itr < maxIter) do
 4:         $th$ = computeThreshold(s,e)                         ▷ Step I
 5:         for i in $0 \ldots n-1$ do
 6:             Compute $upc_i$                                  ▷ Step II
 7:             if $upc_i > th$ then $e[i] = e[i] \oplus 1$      ▷ Step III
 8:         s = $H(c^T + e^T)$                                   ▷ Step IV
 9:         itr = itr + 1
10:     if itr = maxIter then
11:         return $\perp$
12:     else
13:         return e
```

The Black-Gray decoder (in the additional code [7]) and the Backflip decoder [3] use a more complex approach. Similar to the Simple-Parallel decoder, they operate on the error bits in parallel. However, they add a step that re-flips the error bits according to some estimation.

The "`while`" loop of an iteration of the Black-Gray decoder consists of: 1) Perform 1 iteration of the Simple-Parallel decoder and define some bits position candidates that should be reconsidered (i. e., bits that were mistakenly flipped). Then, split them into two lists (black, gray); 2) Reevaluate the bits in the black list, flip them according to the evaluation. Then, recalculate the syndrome; 3) Reevaluate the bits in the gray list, and flip according to the evaluation. Then, recalculate the syndrome.

The Backflip decoder has the following steps: 1) Perform 1 iteration of the Simple-Parallel decoder. For every flipped bit assign a value k. This value indicates that if the algorithm does not end after k iterations, this bit should be flipped back; 2) Flip some bits back according to their k values.

The Backflip$^+$ is variant of Backflip that uses a fixed number iterations as explained in Sect. 1. Technically, the difference is that the condition on the weight of s is moved from the `while` loop to the `if` statement (line 10). This performs the appropriate number of mock iterations.

Remark 1. The decoders use the term iterations differently. For example, one iteration of the Black-Gray decoder is somewhat equivalent to three iterations of the Simple-Parallel decoder. The iteration of the One-Round decoder consists of multiple (not necessarily fixed) "internal" iterations. Comparison of the decoders needs to take this information into account. For example, the performance is determined by the number of iterations times the latency of an iteration, not just by the number of iterations.

3 Idealized Schemes and Concrete Instantiations

We discuss some subtleties related to the requirements from a concrete algorithm in order to be acceptable as substitute for an ideal primitive, and the relation to a concrete implementation.

Cryptographic schemes are often analyzed in a framework where some of the components are modeled as ideal primitives. An ideal primitive is a black-box algorithm that performs a defined flow over some (secret) input and communicates the resulting output (and nothing more). A concrete instantiation of the scheme is the result of substituting the ideal primitive(s) with some specific algorithm(s). We require the following property from the instantiation to consider it *acceptable*: the algorithm should be *possible* to implement without communicating more information than the expected output. From the practical viewpoint, this implies that the algorithm *could be* implemented in constant-time. Note that a specific implementation of an acceptable instantiation of a provably secure scheme can still be insecure (e.g., due to side channel leakage). Special care is needed for algorithms that run with a variable number of steps.

Remark 2. A scheme can have provable security but this does not imply that every instantiation inherits the security properties guaranteed by the proof, or that there even exists an instantiation that inherits them, and an insecure instantiation example does not invalidate the proof of the idealized scheme. For example, an idealized KEM can have an IND-CCA secure proof when using a "random oracle" ideal primitive. An instantiation that replaces the random oracle with a non-cryptographic hash function does not inherit the security proof, but it is commonly acceptable to believe that an instantiation with SHA256 does.

Algorithms with a Variable Number of Steps. Let $\mathcal{A}$ be an algorithm that takes a secret input in and executes a flow with a variable number of steps/iterations $v(in)$ that depends on in. It is not necessarily possible to implement $\mathcal{A}$ in constant-time. In case ("limited") that there is a public parameter b such that $v(in) \leq b$ we can define an equivalent algorithm ($\mathcal{A}^+$) that runs in exactly b iterations: $\mathcal{A}^+$ executes the $v(in)$ iterations of $\mathcal{A}$ and continues with some $b - v(in)$ identical mock iterations. With this definition, we can assume that it is possible to implement $\mathcal{A}^+$ in constant-time. Clearly, details must be provided, and such an implementation needs to be worked out. This could be a challenging task.

Suppose that $v(in)$ is unlimited, i. e., there is no (a-priori) parameter b such that $v(in) \leq b$ (we call this case "unlimited"). It is possible to set a constant parameter b^* and an algorithm $\mathcal{A}^+$ with exactly b^* iterations, such that it emits a failure indication if the output is not obtained after exhausting the b^* iterations. It is possible to implement $\mathcal{A}^+$ in constant-time, but it is no longer equivalent to $\mathcal{A}$, due to the nonzero failure probability. Thus, analysis of $\mathcal{A}^+$ needs to include the dependency of the failure probability on b^*, and consider the resulting implications. Practical considerations would seek the smallest b^* for which the upper bound on the failure probability is satisfactory. Obviously, if $\mathcal{A}$ has originally some nonzero failure probability, then $\mathcal{A}^+$ has a larger failure probability.

Suppose that a cryptographic scheme relies on an ideal primitive. In the limited case an instantiation that substitutes $\mathcal{A}$ (or $\mathcal{A}^+$) is acceptable. However, in the unlimited case, substituting the primitive $\mathcal{A}$ with $\mathcal{A}^+$ is more delicate, due to the failure probability that is either introduced or increased. We summarize the unlimited case as follows.

- To consider $\mathcal{A}$ as an acceptable ideal primitive substitute, $v(in)$ needs to be considered as part of its output, and the security proof should take this information into consideration. Equivalently, the incremental advantage that an adversary can gain from learning $v(in)$ needs to be added to the adversary advantage of the (original) proof.
- Considering $\mathcal{A}^+$ as an acceptable ideal primitive substitute, requires a proof that it has all the properties of the ideal primitive used in the original proof (in particular, the overall failure probability).

Example 1. Consider the IND-CPA RSA PKE. Its model proof relies on the existence of the ideal primitive MODEXP for the decryption (MODEXP (a, x, N) = $a^x \pmod{N}$, where x is secret). Suppose that a concrete instantiation substitutes MODEXP with the square-and-multiply algorithm ($S\&M$). $S\&M$ has a variable number of steps, $t = \mathit{bitlength}(x) + \mathit{wt}(x)$ (modular multiplications), that depends on (secret) x, where $\mathit{wt}(t) \leq \mathit{bitlength}(x)$ is the Hamming weight of x. By definition, in RSA PKE we have that $x < \phi(N) < N$ so x is a-priori bounded and consequently the number of steps in $S\&M$ is bounded by $t \leq 2 \cdot \mathit{bitlength}(x) < 2 \cdot \mathit{bitlength}(N)$. It is easy to define an equivalent algorithm ($S\&M^+$) that runs in exactly $2 \cdot \mathit{bitlength}(N)$ steps by adding mock operations. An instantiation that substitutes $S\&M$ (through $S\&M^+$) for MODEXP can therefore be considered acceptable (up to the understanding of how to define mock steps). This is independent of the practical security of an implementation of RSA PKE instantiated with $S\&M^+$. Such an implementation needs to account for various (side-channel) leaks e.g., branches and memory access patterns. These considerations are attributed to the implementation rather than to the instantiation, because we can trust the possibility to build such a safe implementation.

Application to BIKE. The IND-CCA security proof of BIKE relies on the existence of a decoder primitive that has a negligible DFR. This is a critical decoder's property that is used in the proof. The concrete BIKE instantiation substitutes the idealized decoder with the Backflip decoding algorithm. Backflip has the required negligible DFR. By its definition, Backflip runs in a variable number of steps (iterations) that depends on the input and on the secret key (this property is built into the algorithm's definition).

It is possible to use Backflip in order to define Backflip$^+$ decoder that has a fixed number of steps: a) Fix a number of iterations as a parameter X_{BF}; b) Follow the original Backflip flow but always execute X_{BF} iterations in a way that: if the errors vector (e) is extracted after $Y < X_{BF}$ iterations, execute additional $(X_{BF} - Y)$ identical mock iterations that do not change e; c) After the X_{BF} iterations are exhausted, output a success/failure indication and e on success or a random vector of the expected length otherwise. The difficulty is that the DFR of Backflip$^+$ is a function of X_{BF} (and r) and it may be larger from the DFR of Backflip that is critical for the proof.

It is not clear from [3,20] whether the Backflip decoder is an example of the limited or the unlimited case, but we choose to assume the limited case, based on the following indications. Backflip is defined in [3, Algorithm 4] and the definition is followed by the comment: "The algorithm takes as input [...] and, if it stops, returns an error [...] with high probability, the algorithm stops and returns e". This comment suggests the unlimited case. Here, it is difficult to accept it as a substitution of the ideal primitive, and claim that the IND-CCA security proof applies to this instantiation. In order to make Backflip an ideal primitive substitute, the number of executed steps needs to be considered as part of its output as well. As an analogy, consider a KEM where the decapsulation has nonzero failure probability. Here, an IND-CCA security proof cannot simply rely on the (original) Fujisaki-Okamoto transformation [9], because this would model an ideal decapsulation with no failures. Instead, it is possible to use the $FO^{\not\perp}$ transformation suggested in [12] that accounts for failures. This is equivalent to saying that the modeled decapsulation outputs a shared key and a success/fail indication. Indeed, this transformation was used in the BIKE CCA proof.

On the other hand, we find locations in [3], that state: "In all variants of BIKE, we will consider the decoding as a black box running in bounded time" (Sect. 2.4.1 of [3]) and "In addition, we will bound the running time (as a function of the block size r) and stop with a failure when this bound is exceeded" (Sect. 1.3 of [3]). No bounds and dependency on r are provided. However, if we inspect the reference code [3], we can find that the code sets a *maximal* number of Backflip iterations to 100 (no explanation for this number is provided and this constant is independent of r). Therefore, we may choose to interpret the results of [3,20] as if the 2^{-128} DFR was obtained from simulations with this $X_{BF} = 100$ bound[5], although this is nowhere stated and the simulation data and the derivation of the DFR are also not provided (the reference code operates with $X_{BF} = 100$). With this, it is reasonable to hope that if we take Backflip$^+$ and set $X_{BF} = 100$ we would get a DFR below 2^{-128} and this makes BacklFlip with $X_{BF} = 100$ an acceptable instantiation of an IND-CCA secure BIKE (for the studied r values).

[5] See discussion with some extrapolation methodologies in Appendix C.

The challenge with this interpretation is that the instantiation (Backflip$^+$ and $X_{BF} = 100$) would be impractical from the performance viewpoint. Our paper solves this by showing acceptable instantiations with a much smaller values of X_{BF}. Furthermore, it also shows that there are decoders with a fixed number of iterations that have better performance at the same DFR level.

Implementation. In order to be used in practice, an IND-CCA KEM should have a proper *instantiation* and *also* a constant-time *implementation* that is secure against side-channel attacks (e.g., [8]). Such attacks were demonstrated in the context of QC-MDPC schemes, e.g., the GJS reaction attack [11] and several subsequent attacks [8,15,19]. Other reaction attacks examples include [18] for LRPC codes and [22] for attacking the repetition code used by the HQC KEM [2]. This problem is significantly aggravated when the KEM is used with static keys (e.g., [5,8]).

4 Estimating the DFR of a Decoder with a Fixed Number of Iterations

The IND-CCA BIKE proof assumes a decapsulation algorithm that invokes an ideal decoding primitive. Here, the necessary condition is that the decapsulation has a negligible DFR, e.g., 2^{-128} [3,12]. Therefore, a technique to estimate the DFR of a decoder is an essential tool.

The Extrapolation Method of [20]. An extrapolation method technique for estimating the DFR is shown in [20]. It consists of the following steps: a) Simulate proper encapsulation and decapsulation of random inputs for small block sizes (r values), where a sufficiently large number of failures can be observed; b) Extrapolate the observed data points to estimate the DFR for larger r values.

The DFR analyses in [20] and [3] applies this methodology to decoders that have some maximum number of iterations X_{BF} (we choose to assume that $X_{BF} = 100$ was used). In our experiments Backflip$^+$ always succeeds/fails before reaching 100 iterations for the relevant values of r. Practically, it means that setting $X_{BF} = 100$ can be considered equivalent to setting an unlimited number of iterations.

Our goal is to estimate the DFR of a decoder that is allowed to perform exactly X iterations (where X is predefined). We start from small values of X (e.g., $X = 2, 3, \ldots$) and increase it until we no longer see failures (in a large number of experiments) caused by exhausting X iterations. Larger values of X lead to a smaller DFR.

We tested BIKE-1 and BIKE-3 in Level-1 and Level-3 with the Black-Gray and the Backflip$^+$ decoders. In order to acquire a sufficient number of data points we tested multiple prime r values such that $x^r - 1$ is a primitive polynomial [3]. The specific values are listed in Appendix D.

For our study, we used a slightly different extrapolation method. For every combination (scheme, level, decoder, r) we ran $N_{exp} = 48,000,000$ experiments as follows: a) Generate, uniformly at random, a secret key and an errors vector (e), compute the public key, and perform encapsulation-followed-by-decapsulation (with e); b) Allow the decoder to run up to $X = 100$ iterations[6]; c) Record the actual number of iterations that were required in order to recover e. If the decoder exhausts the 100 iterations it stops and marks the experiment as a decoding failure. For every $X < 100$ we say that the X-DFR is the sum of the number of experiments that fail (after 100 iterations) plus the number of experiments that required more than X iterations divided by N_{exp}. Next, we fix the scheme, the level, the decoder, and X, and we end up with an X-DFR value for every tested r. Subsequently, we perform linear/quadratic extrapolation on the data and receive a curve. We use this curve to find the value r_0 for which the X-DFR is our target probability p_0 and use the pair (r_0, p_0) as the BIKE scheme parameters.

We target three p_0 values: a) $p_0 = 2^{-23} \approx 10^{-7}$ that is reasonable for most practical use cases (with IND-CPA schemes); b) $p_0 = 2^{-64}$ also for an IND-CPA scheme but with a much lower DFR; c) $p_0 = 2^{-128}$, which is required for an IND-CCA Level-1 scheme. The linear/quadratic functions and the resulting r_0 values are given in Appendix E [Table 4].

Our Extrapolation Methodology. In most cases, we were able to confirm the claim of [20] that the evolution of the DFR as a function of r occurs in two phases: quadratic initially, and then linear. As in [20], we are interested in extrapolating the linear part because it gives a more conservative DFR approximation. We point out that the results are sensitive to the method used for extrapolation (see details in Appendix C). Therefore, it is important to define it precisely so that the results can be reproduced and verified. To this end, we determine the starting point of the linear evolution as follows: going over the different starting points, computing the fitting line and picking the one for which we get the best fit to the data points. Here, the merit of the experimental fit is measured by the L2 norm (i. e., mean squared error). The L2 norm is a good choice in our case, where we believe that the data may have a few outliers.

5 Results

A description of the Backflip$^+$ constant-time implementation is provided in Appendix B.

The Experimentation Platform. Our experiments were executed on an AWS EC2 `m5.metal` instance with the 6^{th} Intel® Core^{TM} Generation (Micro Architecture Codename "Sky Lake" [SKL]) Xeon® Platinum 8175M CPU 2.50 GHz. It has 384 GB RAM, 32K L1d and L1i cache, 1MiB L2 cache, and 32MiB L3 cache, where the Intel® Turbo Boost Technology was turned off.

The Code. The core functionality was written in x86−64 assembly and wrapped by assisting C code. The code uses the `PCLMULQDQ`, `AES-NI` and the `AVX2` and

[6] Recall that different decoders have different definition for the term "iterations", see Sect. 2.3.

AVX512 instructions. The code was compiled with gcc (version 7.4.0) in 64-bit mode, using the "O3" Optimization level, and run on a Linux (Ubuntu 18.04.2 LTS) OS. It uses the NTL library [21] compiled with the GF2X library [17].

Figure 8 in Appendix F shows the simulation results for BIKE-1, Level-1 and Level-3, using the Black-Gray and Backflip$^+$ decoders. Note that we use the IND-CCA flows. The left panels present linear extrapolations and the right panels present quadratic extrapolations. The horizontal axis measures the block size r in bits, and the vertical axis shows the simulated $log_{10}(DFR)$ values. Every panel displays several graphs associated with different X values. The minimal X is chosen so that the extrapolated r value for $DFR = 2^{-128}$ is still considered to be secure according to [3]. The maximal value of X is chosen to allow a meaningful extrapolation. We give two examples:

Example 2. Consider Black-Gray. Typically, there exists some number of iterations $j < X_{BG}$, where if decoding a syndrome requires more than j then the decoder fails (w.h.p) even if a large number of iterations X_{BG} is allowed.

The quadratic approximations shown in Fig. 8 yield a nice fit to the data points. However, we prefer to use the more pessimistic linear extrapolation in order to determine the target r.

Validating the Extrapolation. We validated the extrapolated results for every extrapolation graph. We chose some r that is not a data point on the graph (but is sufficiently small to allow direct simulations). We applied the extrapolation to obtain an estimated DFR value. Then, we ran the simulation for this value of r and compared the results. Table 2 shows this comparison for several values of r and the Black-Gray decoder with $X_{BG} = 3$. We note that for $10,267$ and $10,301$ we tested at least 960 million and 4.8 billion tests respectively. In case of $10,301$ decoding always succeeded after $X_{BG} = 4$ iterations, while for $10,267$ there were too few failures for meaningful computation of the DFR. Therefore, we use $X_{BG} = 3$ in our experimentation in order to observe enough failures. For example, the extrapolation for the setting (BIKE-1, Level-1, Black-Gray, $10,301$) estimates 3-DFR $= 10^{-7.55}$ this is very close to the experimented DFRs $10^{-7.56}$.

Table 2. Validating the extrapolation results for the Black-Gray decoder with $X_{BG} = 3$ over two values of r.

r	Extrapolated DFR	Experimented DFR	Number of tests
$10,267$	$10^{-7.13}$	$10^{-7.26}$	$9.6e8$
$10,301$	$10^{-7.55}$	$10^{-7.56}$	$4.8e9$

5.1 Extensive Experimentation

To observe that the Black-Gray decoder does not fail in practice with $r = 11,779$ (i. e., the recommended r for the Backflip decoder) we run extensive simulations. We executed $10^{10} \approx 2^{33}$ tests that generate a random key, encapsulate a message and decapsulate the resulting ciphertext. Indeed, we did not observe any decoding failure (as expected).

5.2 Performance Studies

The performance measurements reported hereafter are measured in processor cycles (per single core), where lower count is better. All the results were obtained using the same measurement methodology, as follows. Each measured function was isolated, run 25 times (warm-up), followed by 100 iterations that were clocked (using the `RDTSC` instruction) and averaged. To minimize the effect of background tasks running on the system, every experiment was repeated 10 times, and the minimum result was recorded.

For every decoder, the performance depends on: a) X - the number of iterations; b) the latency of one iteration. Recall that comparing *just* the number of iterations is meaningless. Table 3 provides the latency ($\ell_{decoder,r}$) of one iteration and the overall decoding latency ($l_{decoder,r,i} = X_{decoder} \cdot \ell_{decoder,r}$) for the Black-Gray and the Backflip$^+$ decoders, for several values of r. The first four rows of the table report for the value $r = 10,163$ that corresponds to the BIKE-1-CPA proposal, and for the value $r = 11,779$ that corresponds to the BIKE-1-CCA proposal. The following rows report values of r for which the decoders achieve the same DFR.

Clearly, the constant-time Black-Gray decoder is faster than the constant-time Backflip$^+$ decoder (when both are restricted to a given number of iterations).

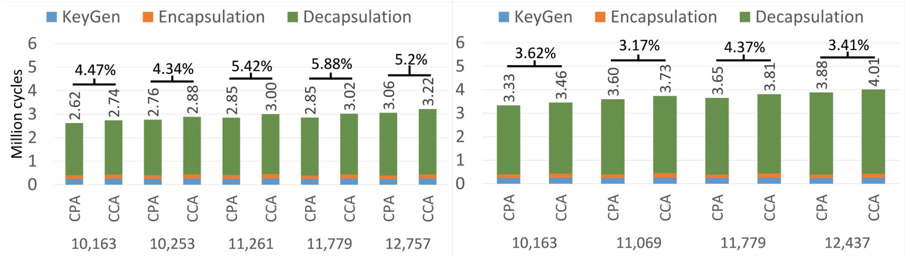
We now compare the performance of the BIKE-1-CCA flows to the performance of the BIKE-1-CPA flows, for given r values, using the Black-Gray decoder with $X_{BG} = 3, 4$. Note that values of r that lead to DFR $> 2^{-128}$ cannot give IND-CCA security. Furthermore, even with BIKE-1-CCA flows and r such that DFR $\leq 2^{-128}$, IND-CCA security is not guaranteed (see the discussion in Sect. 6). The results are shown in Fig. 2. The bars show the total latency of the key generation (blue), encapsulation (orange), and decapsulation (green) operations. The slowdown imposed by using the BIKE-1-CCA flows compared to using the BIKE-1-CPA flows is indicated (in percents) in the figure. We see that the additional cost of using BIKE-1-CCA flows is only ∼6% in the worst case.

6 Weak Keys: A Gap for Claiming IND-CCA Security

Our analysis of the decoders, the new parameters suggestion, and the constant-time implementation makes significant progress towards a concrete instantiation

Table 3. A performance comparison of the Black-Gray and the Backflip$^+$ decoders for BIKE-1 Level-1. The r values were chosen according to Table 4.

DFR	Decoder	r	$X_{decoder}$	$\ell_{decoder,r}$ (cycles)	$l_{decoder,r,i}$ (million cycles)
2^{-19}	Black-Gray	10,163	3	702,785	2.1
2^{-17}	Backflip$^+$	10,163	8	751,246	6.0
2^{-101}	Black-Gray	11,779	4	784,903	3.13
2^{-58}	Backflip$^+$	11,779	9	841,806	6.73
2^{-23}	Black-Gray	10,253	3	743,168	2.22
2^{-23}	Black-Gray	10,163	4	702,785	2.8
2^{-23}	Backflip$^+$	10,499	8	777,478	6.22
2^{-23}	Backflip$^+$	10,253	9	764,959	6.88
2^{-64}	Black-Gray	11,261	3	769,212	2.3
2^{-64}	Black-Gray	11,003	4	769,820	3.0
2^{-64}	Backflip$^+$	12,781	8	907,905	7.26
2^{-64}	Backflip$^+$	12,011	9	856,084	7.7
2^{-128}	Black-Gray	12,781	3	849,182	2.54
2^{-128}	Black-Gray	12,347	4	841,310	3.36
2^{-128}	Backflip$^+$	14,797	9	1,024,798	9.22



(a) $X_{BG} = 3$

(b) $X_{BG} = 4$

Fig. 2. Comparison of BIKE-1-CPA flows and BIKE-1-CCA flows, running with the Black-Gray decoder and $X_{BG} = 3, 4$ for several values of r: $r = 10,163$ the original BIKE-1-CPA; $r = 11,779$ the original BIKE-1-CCA; r values that correspond to DFR of $2^{-23}, 2^{-64}, 2^{-128}$, according to Table 4. Note that values of r that lead to $DFR > 2^{-128}$ do not give IND-CCA security. The vertical axis measures latency in millions of cycles (lower is better). The additional cost of using the IND-CCA flows it at most 6%.

and implementation of IND-CCA BIKE. However, we believe that there is still a subtle missing gap for claiming IND-CCA security, that needs to be addressed.

The remaining challenge is that a claim for IND-CCA security depends on having an underlying δ-correct PKE (for example with $\delta = 2^{-128}$ for Level-1) [12]. This notion is different from having a DFR of 2^{-128}, and leads to the following problem. The specification [3] defines the DFR as "the probability for the decoder to fail when the input (h_0, h_1, e_0, e_1) is distributed uniformly". The δ-correctness property of a PKE/KEM is defined through Eqs. (1), (2) above. These equations imply that δ is the average of the *maximum* failure probability taken over all the possible messages. By contrast, the DFR notion relates to the average probability.

Remark 3. We also suggest to fix a small inaccuracy in the statement of the BIKE-1 proof [3]: "... the resulting KEM will have the exact same DFR as the underlying cryptosystem ...". Theorem 3.1 of [12] states that: "If PKE is δ-correct, then PKE$_1$ is $\delta 1$-correct in the random oracle model with $\delta 1(qG) = qG \cdot \delta$.[...]". Theorem 3.4 therein states that: "If PKE$_1$ is $\delta 1$-correct then KEM$^{\not\perp}$ is $\delta 1$-correct in the random oracle model [..]"[7]. Thus, even if DFR $= \delta$, the statement should be "the resulting KEM is $(\delta \cdot qG)$-correct, where the underlying PKE is δ-correct".

To illustrate the gap between the definitions, we give an example for what can go wrong.

Example 3. Let $\mathcal{S}$ be the set of valid secret keys and let $|\mathcal{S}|$ be its size. Assume that a group of weak keys $\mathcal{W}$ exists[8], and that $\frac{|\mathcal{W}|}{|\mathcal{S}|} = \bar{\delta} > 2^{-128}$. Suppose that for every key in $\mathcal{W}$ there exists at least one message for which the probability in Eq. (1) equals 1. Then, we get that $\delta > \bar{\delta} > 2^{-128}$. By comparison the average DFR can still be upper bounded by 2^{-128}. For instance, let $|\mathcal{S}| = 2^{130}$, $|\mathcal{W}| = 2^4$ and let the failure probability over all messages for every weak key be 2^{-10}. Let the failure probability of all other keys be 2^{-129}. Then,

$$\begin{aligned} DFR &= Pr(\texttt{fail} \mid k \in \mathcal{W}) \cdot Pr(k \in \mathcal{W}) + Pr(\texttt{fail} \mid k \in \mathcal{S} \setminus \mathcal{W}) \cdot Pr(k \in \mathcal{S} \setminus \mathcal{W}) \\ &= \frac{|\mathcal{W}|}{|\mathcal{S}|} \cdot 2^{-10} + \frac{|\mathcal{S}| - |\mathcal{W}|}{|\mathcal{S}|} \cdot 2^{-129} \\ &= 2^{-126} \cdot 2^{-10} + (1 - 2^{-126}) \cdot 2^{-129} \\ &= 2^{-136} + 2^{-129} - 2^{-255} < 2^{-128} \end{aligned}$$

6.1 Constructing Weak Keys

Currently, we are not aware of studies that classify weak keys for QC-MDPC codes or bound their numbers. To see why this gap cannot be ignored we designed a series of tests that show the existence of a relatively large set of weak keys. Our examples are inspired by the notion of "spectrum" used in [8,11,15]. To construct the keys we changed the BIKE key generation. Instead of generating a random h_0, we start by setting the first $f = 0, 20, 30, 40$ bits, and then select randomly the positions of the additional $(d - f)$ bits. The generation of h_1 is unchanged.

[7] Here, KEM$^{\not\perp}$ refers to a KEM with implicit rejection, and qG is the number of invocation of the random oracle G (H in the case of BIKE-1).

[8] Our definition of weak keys is different form that of [4], where a weak key is a secret key that can be exposed from the public key alone.

Since it is difficult to observe failures and weak keys behavior when r is large, we study $r = 10,163$ (of BIKE-1 CPA) and also $r = 9,803$ that amplify the phenomena.

Figure 7 in Appendix F shows the behavior of the Black-Gray decoder for $r = 9,803$ and $r = 10,163$ with $f = 0, 20, 30, 40$ after $X_{BG} = 1, 2, 3, 4$ iterations. In every case (Panel) we choose randomly $10,000$ keys. For every key we choose randomly $1,000$ errors vectors. The histograms show the weight of an "ideal" errors vector e after the X_{BG} iteration (horizontal axis). We see that as f grows the number of un-decoded error bits after every iteration increases. For $f \leq 30$, decoding often succeeds after 3 iterations. However, for $f = 40$ it is possible to decode the error after 4 iterations only when $r = 10,163$, but not for $r = 9,803$. In other words, if we fix $X_{BG} = 3$ for the Black-Gray decoder and use $r = 9,803$ we see ∼100% failures with weak keys defined by $f = 40$. This shows that for a given decoder the set of weak keys depends on r and X.

Remark 4 (Other decoders). Figure 7 shows how the weak keys impact the decoding success probability for chosen r and X_{BG} with the Black-Gray decoder. Note that such results depend on the specific decoder. To compare, Backflip$^+$ calculates the unsatisfied parity checks threshold in a more conservative way, and therefore requires more iterations. Weak keys would lead to a different behavior. When we repeat our experiment with the Simple-Parallel decoder, we see that almost all tests fail even with $f = 19$.

Figure 3 shows additional results with the Black-Gray decoder and $r = 9,803$. Panel (a) shows the histogram for $f = 0$ (i. e., reducing to the standard h_0 generation), and Panel (b) shows the histogram for $f = 30$. The horizontal axis measures the number of failures x out of $10,000$ random errors. The vertical axis counts the number of keys that have a DFR of $x/10,000$. For $f = 0$, the average and standard deviation are $\mathbb{E}(x) = 119.06$, and $\sigma(x) = 10.91$. However, when $f = 30$, the decoder fails much more often and we have $\mathbb{E}(x) = 9,900.14$, and $\sigma(x) = 40.68$. This shows the difference between the weak keys and the "normal" randomized keys and that the DFR cannot be predicted by the "average-case" model. It is also interesting to note that for $f = 30$ we do not get a Gaussian like distribution (unlike the histogram with $f = 0$).

The remaining question is: what is the probability to hit a weak key when the keys are generated randomly as required? Let $\mathcal{W}_f$ be the set of weak keys that correspond to a given value of f. Define $z_{r,f}$ as the relative size of $\mathcal{W}_f$. Then

$$z_{r,f} = \frac{|\mathcal{W}_f|}{|\mathcal{S}|} = \frac{\binom{r-f}{d-f}}{\binom{r}{d}} \tag{3}$$

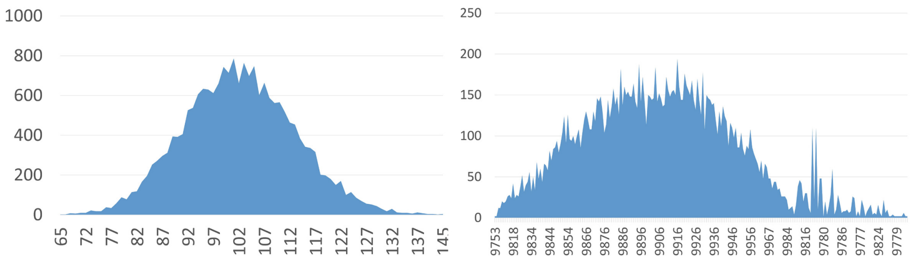
Note that choosing a larger f decreases the size of $\mathcal{W}_f$, i. e., if $f_2 < f_1$ then $\mathcal{W}_{f_1} \subseteq \mathcal{W}_{f_2}$. It is easy to compute that

$$z_{9803,0} = z_{10163,0} = 1$$

$$z_{9803,10} = 2^{-72}, \quad z_{9803,20} = 2^{-146}, \quad z_{9803,30} = 2^{-223}, \quad z_{9803,40} = 2^{-304},$$

$$z_{10163,10} = 2^{-72}, \quad z_{10163,20} = 2^{-147}, \quad z_{10163,30} = 2^{-225}, \quad z_{10163,40} = 2^{-306}$$

The conclusion is that while the set $\mathcal{W}_f$ is large, its relative size (from the set of all keys) is still below 2^{-128}. Therefore, this construction *does not* show that BIKE-1 after our fix is necessarily *not* IND-CCA secure. However, it clearly shows that the problem cannot be ignored, and the claim that BIKE *is* IND-CCA secure requires further justification. In fact, there are other patterns and combinations that give sets of weak keys with a higher relative size (e.g., setting every other bit of h_0, f times). We point out again that any analysis of weak keys should relate to a *specific* decoder and a *specific* choice of r.



(a) (b)

Fig. 3. Black-Gray decoder, $r = 9,803$ with $f = 0$ (Panel (a)) and $f = 30$ (Panel (b)). The horizontal-axis measures the number of failures x out of $10,000$ random errors vectors. The vertical-axis counts the number of keys that have a DFR of $x/10,000$. The conclusion is that there are keys that lead to higher DFR.

7 Discussion

The Round-2 BIKE [3] represents significant progress in the scheme's design, and offers an IND-CCA version, on top of the IND-CPA KEM that was defined in Round-1. This paper addresses several difficulties and challenges and solves some of them.

- The Backflip decoder runs in a variable number of steps that depends on the input and the secret key. We fix this problem by defining a variant, Backflip$^+$, that, by definition, runs X_{BF} iterations for a parameter X_{BF}. We carry out the analysis to determine the values of X_{BF} where Backflip$^+$ has DFR of 2^{-128}, and provide all of the details that are needed in order to repeat and reproduce our experiments. Furthermore, we show that for the target DFR, the values of X_{BF} are relatively small e.g., 12. (much less than 100 as implied for Backflip).
- Inspired by the extrapolation method suggested in [20], we studied the Black-Gray decoder (already used in Round-1 Additional code [7]) that we defined to have a fixed number of steps (iterations) X_{BG}. Our goal was to find values of X_{BG} that guarantee the target DFR for a given r. We found that the values

of r required with Black-Gray are smaller than the values with Backflip$^+$. It seems that achieving the low DFR (2^{-128}) should be attributed to increasing r, independently of the decoding algorithm. The ability to prove this for some decoders is attributed to the extrapolation method.

- After the decoders are defined to run a fixed number of iterations, we could build constant-time software implementations (with memory access patterns and branches that reveal no secret information). This is nowadays the standard expectation from cryptographic implementations. We measured the resulting performance (on a modern x86-64 CPU) to find an optimal "decoder-X-r" combination. Table 3 shows that for a given DFR, the Black-Gray decoder is always faster than Backflip$^+$.
- The analysis in Sect. 6 identifies a gap that needs to be addressed in order to claim IND-CCA security for BIKE. It relates to the difference between average DFR and the δ-correctness definition [12]. A DFR of (at most) 2^{-128} is a necessary requirement for IND-CCA security, which BIKE achieves. However, it is not necessarily sufficient. We show how to construct a "large" set of weak keys, but also show that it still not sufficiently large to invalidate the necessary δ-correctness bound. This is a positive indication, although there is no proof that the property is indeed satisfied. This gap remains as an interesting research challenge to pursuit. The problem of bounding (or eliminating) the number of weak keys is not specific to BIKE. It is relevant for other schemes that claim IND-CCA security and their decapsulation has nonzero failure probability. With this, we can state the following.

BIKE-1-CCA, instantiated with Black-Gray (or Backflip$^+$) decoder with the parameters that guarantee DFR of 2^{-128}, and with the accompanying constant-time implementation, is IND-CCA secure, under the assumption that its underlying PKE is 2^{-128}-correct.

7.1 Methodologies

Our performance measurements were carried on an x86-64 architecture. Studies on different architectures can give different results we therefore point to an interesting study of the performance of other constant-time decoders on other platforms [14]. Note that [14] targets schemes that use ephemeral keys with relatively large DFR and only IND-CPA security.

Differences in the DFR Estimations. Our DFR prediction methodology may be (too) conservative and therefore yields more pessimistic results than those of [3]. One example is the combination (BIKE-1, Level-1, Backflip$^+$ decoder, $r = 11,779$, $X_{BF} = 10$). Here, [3] predicts a 100−DFR of 2^{-128} and our linear extrapolation for the 10-DFR predicts only 2^{-71}($\approx 10^{-21}$). To achieve a 10-DFR of 2^{-128} we need to set $r = 13,892(>11,779)$. Although the Backflip$^+$ decoder with $X_{BF} = 10$ is not optimal, it is important to understand the effect of different extrapolations. Comparing to [3,20] is difficult (no information that allows us to repeat the experiments), we attempt to provide some insight by acquiring data points for Backflip$^+$ with $X_{BF} = 100$ and applying our extrapolation methodology. Indeed, the results we obtain are still more pessimistic, but if we apply a different extrapolation methodology ("Last-Linear") we get closer to [3]. The details are given in Appendix C.

Another potential source of differences is that Backflip has a recovery mechanism (TTL). For Backflip$^+$ this mechanism is limited due to setting $X_{BF} \leq 11$. It may be possible to tune Backflip and Backflip$^+$ further by using some fine-grained TTL equations that depend on r. Information on the equations that were used for [3] was not published, so we leave tuning for further research.

7.2 Practical Considerations for BIKE[9]

Our Decoder Choice. We report our measurements only for Black-Gray and Backflip$^+$ because other decoders that we know either have a worse DFR (e.g., Parallel-Simple) or are inherently slow (e.g., Step-by-Step). Our results suggest that instantiating BIKE with Black-Gray is recommended. We note that the higher number of iterations required by Backflip$^+$ is probably because it uses a more conservative threshold function than Black-Gray.

Recommendations for the BIKE Flows. Currently, BIKE has two options for executing the key generation, encapsulation, and decapsulation flows. One for an IND-CPA KEM, and another (using the $FO^{\not\perp}$ transformation [12]) for an IND-CCA scheme, to deny a chosen ciphertext attack from the encapsulating party. It turns out that the performance difference is relatively small. As shown in Fig. 2 for BIKE-1, the overhead of the IND-CCA flows is less than 6% (on x86-64 platforms). With such a low overhead, we believe that the BIKE proposal could gain a great deal of simplification by using *only* the IND-CCA flows. This is true even for applications that intend to use only ephemeral keys, in order to achieve forward secrecy. Here, IND-CCA security is not mandatory, and IND-CPA security suffice. However, using the $FO^{\not\perp}$ transformation could be a way to reduce the risk of inadvertent repetition ("misuse") of a supposedly ephemeral key, thus buying some multi-user-multi-key robustness. By applying this approach, the scheme is completely controlled by choosing a single parameter r (with the same implementation). For example, with the Black-Gray decoder and $X_{BG} = 4$, the choice $r = 11,001$ gives DFR $= 2^{-64}$ with competitive performance. A DFR of 2^{-64} is sufficiently low and can be ignored by all practical considerations.

Choosing r. The choice of r and X_{BG} gives an interesting trade-off between bandwidth and performance[10]. A larger value of r increases the bandwidth but reduces the DFR when X_{BG} is fixed. On the other hand, it allows to reduce

[9] The recommendations given here are the opinion of the authors of this paper.

[10] BIKE specification [3, Section 2.4.5] states: "An interesting consequence is that if w and t are fixed, a moderate modification of r (say plus or minus 50%) will not significantly affect the resistance against the best known key and message attacks".

X_{BG} while maintaining the same DFR. This could lead to better performance. We give one example. To achieve DFR $= 2^{-23}$ the choice of $X_{BG} = 4$ and $r = 10,163$ leads to decoding at $2.8M$ cycles. The choice $X_{BG} = 3$ and a slightly larger $r = 10,253$ leads to decoding at $2.22M$ cycles. Complete details are given in Table 3.

General Recommendations for the BIKE Suite. Currently, BIKE [3] consists of 10 variants: BIKE-1 (the simplest and fastest); BIKE-2 (offering bandwidth optimization at the high cost of polynomial inversion); BIKE-3 (simpler security reduction with the highest bandwidth and lowest performance). In addition, there are also BIKE-2-batch and BIKE-3 with bandwidth optimization. Every version comes with two flavors IND-CPA and IND-CCA. On top of this, every option comes with three security levels (L1/L3/L5). Finally, the implementation packages include generic code and optimization for AVX2 and AVX512. We believe that this abundance of options involves too high complexity and reducing their number would be very useful. For Round-3 we recommend to define BIKE as follows: BIKE-1 CCA instantiated with the Black-Gray decoder with $X_{BG} = 3$ iterations. Offer Level-1 with $r = 11,261$ targeting DFR $= 2^{-64}$ and $r = 12,781$ targeting DFR $= 2^{-128}$, as the main variants. In all cases, use ephemeral keys, for forward secrecy. For completeness, offer also a secondary variant for Level-3 with $r = 24,659$ targeting DFR $= 2^{-128}$.

The code that implements these recommendations was contributed to (and already merged into) the open-source library LibOQS [1]. It uses the choice of $r = 11,779$, following the block size of the current Round-2 specification (this choice of r leads to a DFR of 2^{-86}).

Vetting Keys. We recommend to use BIKE with ephemeral keys and forward secrecy. In this case we do not need to rely on the full IND-CCA security properties of the KEM. However, there may be usages that prefer to use static keys. Here, we recommend the following way to narrow the DFR-δ-correctness gap pointed above by "vetting" the private key. For static keys we can assume that the overall latency of the key generation generation phase is less significant. Therefore, after generating a key, it would be still acceptable, from the practical viewpoint, to vet it experimentally. This can be done by running encapsulation-followed-by-decapsulation for some number of trials, in the hope to identify a case where the key is obviously weak. A more efficient way is to generate random (and predefined) errors and invoke the decoder. We point out that the vetting process can also be applied offline.

Acknowledgments. This research was partly supported by: NSF-BSF Grant 2018640; The BIU Center for Research in Applied Cryptography and Cyber Security, and the Center for Cyber Law and Policy at the University of Haifa, both in conjunction with the Israel National Cyber Directorate in the Prime Minister's Office.

A Black-Gray Decoder

Algorithm 2. e=Black-Gray(c, H)

```
Input: Parity-check matrix $H \in \mathbb{F}_2^{r \times n}$, $c \in \mathbb{F}_2^n$, $X_{BG}$  (maximal # of iterations)
Output: The error $e \in \mathbb{F}_2^n$
Exception: "decoding failure" return $\perp$
procedure BLACK-GRAY($c$, $H$)
    $s = Hc^T$, $e = 0$, $\delta = 4$
    $B = \emptyset$, $G = \emptyset$                          ▷ Black and Gray position sets

    for itr in $0 \ldots X_{BG} - 1$ do
        $th$ = computeThreshold(s)
        $upc[n-1:0] = computeUPC(s, H)$
        for i in $0 \ldots n-1$ do                             ▷ Step I
            if $upc[i] \geq th$ then
                $e[i] = e[i] \oplus 1$                         ▷ Flip an error bit
                $B = B \cup i$                                 ▷ Update the Black set
            else if $upc[i] > th - \delta$ then
                $G = G \cup i$                                 ▷ Update the Gray set
        $s = H(c^T + e^T)$                                     ▷ Update the syndrome

        $upc[n-1:0] = computeUPC(s, H)$                        ▷ Step II
        for $b \in B$ do
            if $upc[b] > ((d+1)/2)$ then
                $e[b] = e[b] \oplus 1$                         ▷ Flip an error bit
        $s = H(c^T + e^T)$                                     ▷ Update the syndrome

        $upc[n-1:0] = computeUPC(s, H)$                        ▷ Step III
        for $g \in G$ do
            if $upc[g] > ((d+1)/2)$ then
                $e[g] = e[g] \oplus 1$                         ▷ Flip an error bit
        s $= H(c^T + e^T)$                                     ▷ Update the syndrome

    if $(wt(s) \neq 0)$ then
        return $\perp$
    else
        return e
```

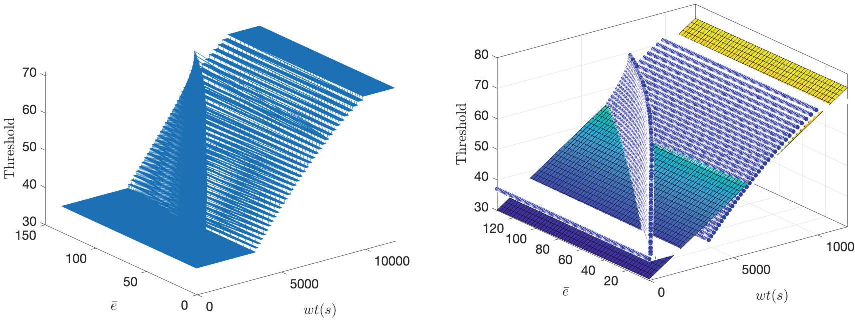
B Implementing Backflip$^+$ in Constant-Time

Here, we show how to define and implement a constant-time Backflip$^+$ decoder, based on a constant-time Black-Gray decoder. The Backflip$^+$ decoder differs from the Black-Gray decoder in two aspects: a) it uses a new mechanism called TTL; b) it uses new equations for calculating the thresholds. The TTL mechanism is a "smart queue" where the decoder flips back some error bits when it believes that they were mistakenly flipped in previous iterations. It does so unconditionally and it can flip bits even after 5 iterations. The Black-Gray decoder uses a different type of TTL, where the black and gray lists serve as the "smart queue". However, the error bits are flipped back after only 1 iteration, conditionally, through checking certain thresholds. Indeed, as we report below the differences are observed in cases where the Black-Gray decoder failed to decode after 4 iterations and then w.h.p fails completely. The Backflip decoder shows better recovery capabilities in such cases. Implementing the new TTL queue in constant-time relies mostly on common constant-time techniques.

Handling the New Threshold Function. The Backflip decoder thresholds are a function of two variables [3][Section 2.4.3]: a) the syndrome weight $wt(s)$ as in the Black-Gray decoder; b) the number of error bits that the decoder believes it flipped (denoted $\bar{e}$). This function outputs higher thresholds compared to the Black-Gray decoder. This is a conservative approximation. We believe that the design of the Backflip decoder tends to avoid flipping the "wrong" bits so that the decoder would have better recovery capabilities and a lower DFR (assuming that it can execute an un-bounded number of iterations). We point out that evaluating the function involves computing logarithms, exponents, and function minimization, and it is not clear how this can be implemented in constant-time (the reference code [3] is not implemented in constant-time).

One way to address this issue is to pre-calculate the finite number of pairs $(wt(s), \bar{e})$ and their function evaluation, store them in a table, and read them from the table in constant-time. This involves very high latencies.

Similarly to the Black-Gray decoder (in BIKE-1-CPA [3]), we approximate the thresholds function - which is here a function of two variables. A first attempt is shown in Fig. 4. We compute the function over all the valid/relevant inputs and then compute an approximation by fitting it to a plane. Unfortunately, this approximation is not sufficiently accurate, an experiment with $r = 11,779$ (as in BIKE-1-CCA [3]) gave an estimated DFR of 10^{-4} .



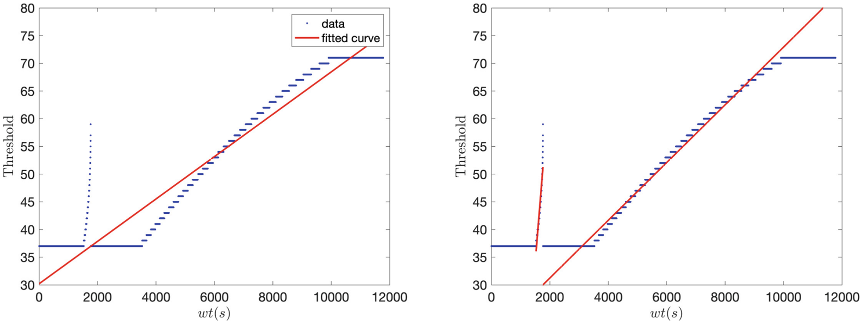
(a) The thresholds function.

(b) Approximating the function (blue) using a plane (yellow,turquoise).

Fig. 4. Approximating the Backflip decoder thresholds function.

To improve the approximation we project the function onto the plane $\bar{e} = e1$ $(0 \leq e1 \leq t)$. Then, for every valid $e1$, we compute the linear approximation and tabulate the coefficients. Figure 5, Panel (a) illustrates the linear approximation for $e1 = 25$. These thresholds improve the DFR but it is still too high.

A refinement can be obtained by partitioning the approximation into five regions. The projection graph in Fig. 5 can be partitioned in five intervals as follows: a) $[a_0, a_1]$, $[a_2, a_3]$, where the threshold is fixed to some minimum value (min); b) $[a_4, a_5]$ where the threshold is d; c) $[a_1, a_2]$ and $[a_3, a_4]$ where the threshold (th) is approximated using $th = b_0 wt(s) + b_1$ and $th = c_0 wt(s) + c_1$, respectively. For $r = 11,779$ the values we use are $a_0 = 0$, $a_1 = 1,578$, $a_2 = 1,832$ $a_3 = 3,526$, $a_4 = 9,910$ $a_5 = r$. The results is shown Fig. 5 Panel (b) for $\bar{e} = 25$. We use these values to define the table (T) with t rows and 8 columns. Every row contains the a_1, a_2, a_3, a_4, b_0, b_1, c_0, c_1 values that correspond to the projection on the plane $\bar{e}$.



(a) Linear approximation of $wt(s) \in [0, r]$ (b) Five linear approximation per part.

Fig. 5. Approximating the threshold function when $\bar{e} = 25$ is fixed.

For every $(s1, e1) = (wt(s), \bar{e})$ the threshold is computed by

```
if   (s1 < T[e1][0]) threshold = min;
elif (s1 < T[e1][1]) threshold = T[e1][4] * s1 + T[e1][5];
elif (s1 < T[e1][2]) threshold = min;
elif (s1 < T[e1][3]) threshold = T[e1][6] * s1 + T[e1][7];
else threshold = d
```

To evaluate the thresholds in constant-time we used a constant-time function `secure_le_mask` that compares two integers j, k and returns the mask `0x0` if $j < k$ and the mask `0xffffffff` otherwise. The threshold computation is now:

```
cond0 = secure_le_mask(T[e1][0], s1)
cond1 = secure_le_mask(T[e1][1], s1) & ~secure_le_mask(T[e1][0], s1)
cond2 = secure_le_mask(T[e1][2], s1) & ~secure_le_mask(T[e1][1], s1)
cond3 = secure_le_mask(T[e1][3], s1) & ~secure_le_mask(T[e1][2], s1)
cond4 = ~secure_le_mask(T[e1][3],s1)

res  = cond0 & min
res += cond1 & round(T[e1][4] * s1 + T[e1][5])
res += cond2 & min
res += cond3 & round(T[e1][6] * s1 + T[e1][7])
res += cond4 & max
return res
```

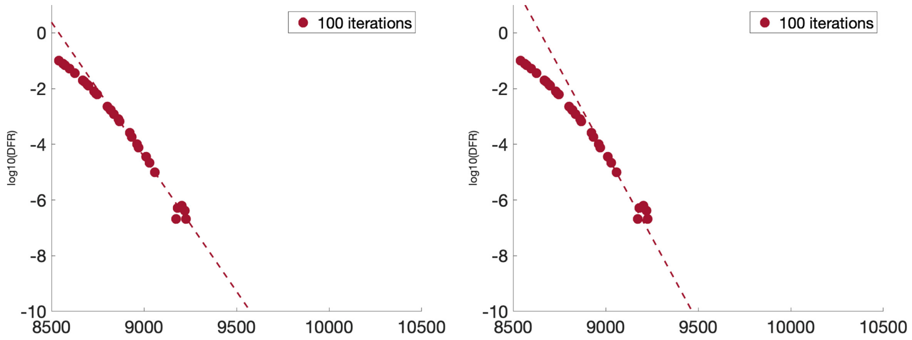
With this we can implement Backflip$^+$ in constant-time, provided that we fix a-priori the number of iterations.

C Achieving the Same DFR Bounds as of [20]

We ran experiments with Backflip$^+$ and $X_{BF} = 100$ for BIKE-1 Level-1, scanning all the 34 legitimate $r \in [8500, 9340]$ (prime r values such that $x^r - 1$ is a primitive polynomial) with $4.8M$ tests for every value. Applying our extrapolation methodology (see Sect. 4) to the acquired data leads to the results illustrated in Fig. 6 Panels (a) and (b). The figure highlights the pairs (DFR; r) for DFR 2^{-64} and 2^{-128} with the smallest possible r. For example, with $r = 12,539$ the linear extrapolation gives DFR of 2^{-128}. Note that [3] claims a DFR of 2^{-128} for a smaller $r = 11,779$. For comparison, with $r = 11,779$ our methodology gives a DFR of 2^{-104}. We can guess that either different TTL values were used for every r, or that other r values were used, or that a different extrapolation methodology was applied.

We show one possible methodology ("Last-Linear") that gives a DFR of $\sim 2^{-128}$ with $r = 11,779$ when applied to the acquired data: a) Ignore the points from the data-set for which 100−DFR is too low to be calculated reliably (e.g., the five lower points in Fig. 6); b) Draw a line through the last two remaining data points with the highest values of r. The rationale is that the "linear regime" of the DFR evolution starts for values of r that are beyond those that can be estimated in an experiment. Thus, a line drawn through two data points where r is smaller than the starting point of the linear regime leads to an extrapolation that is lower-bounded by the "real" linear evolution. With this approach, the question is how to choose the two points for which experimental data is obtained and the DFR is extrapolated from.

This shows that different ways to acquire and interpret the data give different upper bounds for the DFR. Since the extrapolation shoots over a large gap of r values, the results are sensitive to the chosen methodology. It is interesting to note that if we take our data points for Black-Gray and $X_{BG} = 5$ and use the Last-Linear extrapolation, we can find two points that would lead to 2^{-128} and $r = 11,779$, while more conservative methodology gives only 2^{-101}.



(a) lin. ext., our method. (DFR, r) = (2^{-64}; $10,589$), (2^{-128}; $12,539$)

(b) lin. ext. Last-Linear. (DFR, r) = (2^{-64}; $10,253$), (2^{-128}; $11,813$)

Fig. 6. BIKE-1 Level-1 Backflip^{+} different extrapolation methods. See the text for details. The sub-captions detail the (DFR; r) for DFR values: 2^{-64}, 2^{-128}.

D Additional Information

The following values of r were used by the extrapolation method:

- BIKE-1 Level-1: 9349, 9547, 9749, 9803, 9859, 9883, 9901, 9907, 9923, 9941, 9949, 10037, 10067, 10069, 10091, 10093, 10099, 10133, 10139.
- BIKE-1 Level-3: 19013, 19037, 19051, 19069, 19141, 19157, 19163, 19181, 19219, 19237, 19259, 19301, 19333, 19373, 19387, 19403, 19427, 19469, 19483, 19507, 19541, 19571, 19597, 19603, 19661, 19709, 19717, 19739, 19763, 19813, 19853.

E The Linear and the Quadratic Extrapolations

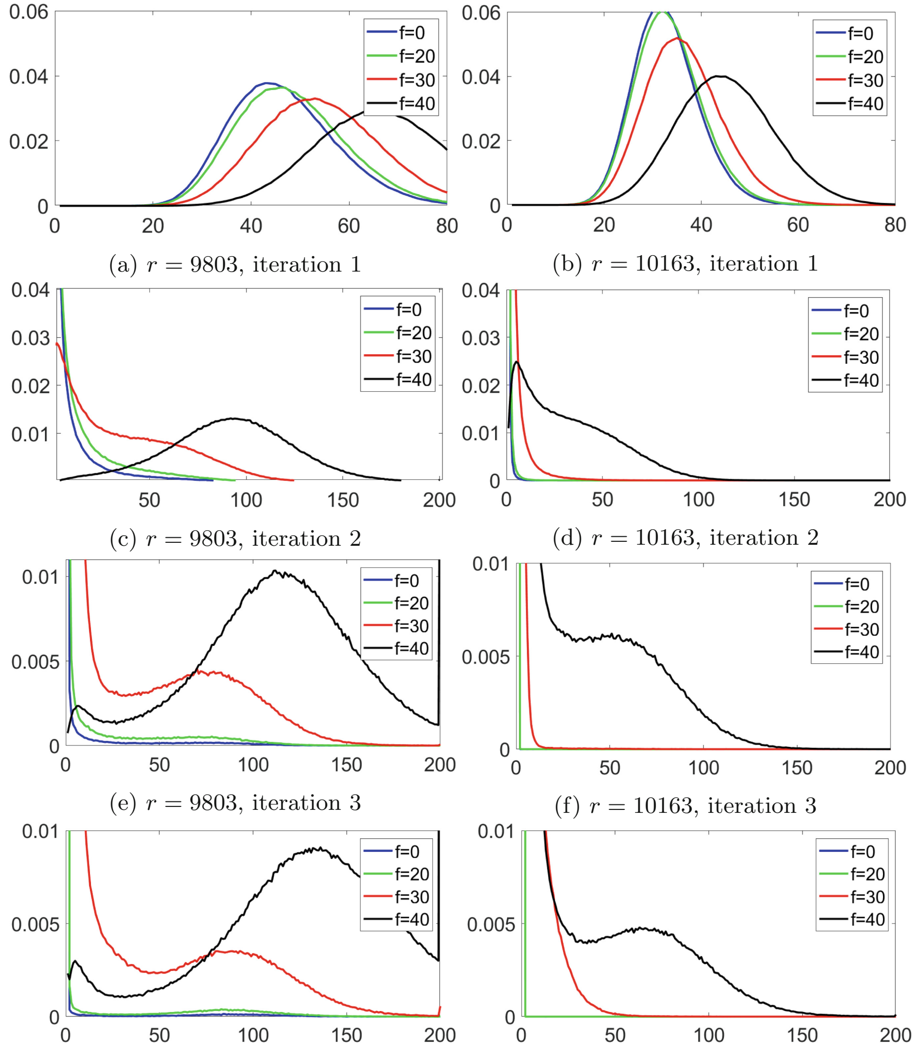
Table 4 gives the equations for the linear and the quadratic extrapolation together with the extrapolated values of r for a DFR of 2^{-23}, 2^{-64}, and 2^{-128}. It covers the tuple (scheme, level, decoder, X), where decoder $\in$ {BG=Black-Gray, BF=Backflip^{+}}.

The BIKE specification [3] chooses r to be the minimum required for achieving a certain security level, and the best bandwidth trade-off. It also indicates that it is possible to increase r by "plus or minus 50%" (leaving w, t fixed) without reducing the complexity of the best known key/message attacks. This is an interesting observation. For example, increasing the BIKE-1 Level-3 $r = 19,853$ by 50% gives $r = 29,779$ which is already close to the BIKE-1 Level-5 that has $r = 32,749$ (of course with different w and t). We take a more conservative approach and restrict r values to be at most 30% above their CCA values stated in [3]. Table 4 labels values beyond this limit as N/A.

Table 4. The linear and the quadratic extrapolation equations, and the computed r values for a given DFR. The cases labeled with N/A are those where the value of r to achieve a target DFR could not be found in the range $[0.7r', 1.3r']$, where r' is the recommended value for IND-CCA security in [3]

KEM	Lev.	Decoder	Iter.	Lin. start	Lin. eq. (a,b) s.t. $\log_{10}\mathtt{DFR} = ar + b =$	2^{-23}	2^{-64}	2^{-128}	Quad. eq. (a, b, c) s.t. $\log_{10}\mathtt{DFR} = ar^2 + br + c =$	2^{-23}	2^{-64}	2^{-128}
BIKE-1	1	BG	3	10	(−1.25e−2, 121)	10,253	11,261	12,781	(−1.05e−5, 1.97e−1, −927)	10,253	10,789	11,317
BIKE-1	1	BG	4	9	(−1.45e−2, 140)	10,163	11,003	12,347	(−1.16e−5, 2.18e−1, −1020)	10,139	10,667	11,197
BIKE-1	1	BG	5	9	(−1.49e−2, 144)	10,133	10,973	12,251	(−1.18e−5, 2.20e−1, −1030)	10,133	10,667	11,171
BIKE-1	1	BF	8	9	(−5.40e−3, 49.8)	10,499	12,781	N/A	(−6.86e−7, 8.11e−3, −16.8)	10,459	12,149	14,107
BIKE-1	1	BF	9	6	(−6.92e−3, 63.8)	10,253	12,011	14,797	(−1.16e−6, 1.62e−2, −50.9)	10,253	11,579	13,109
BIKE-1	1	BF	10	8	(−8.40e−3, 77.6)	10,067	11,549	13,829	(−1.88e−6, 2.90e−2, −108)	10,067	11,197	12,437
BIKE-1	1	BF	11	7	(−1.12e−2, 104)	9,949	11,069	12,781	(−3.41e−6, 5.77e−2, −243)	9,949	10,883	11,867
BIKE-1	3	BG	3	10	(−6.97e−3, 133)	20,051	21,821	24,659	(−2.39e−6, 8.64e−2, −780)	19,997	21,059	22,189
BIKE-1	3	BG	4	10	(−8.70e−3, 166)	19,853	21,269	23,459	(−3.34e−6, 1.22e−1, −1110)	19,813	20,717	21,683
BIKE-1	3	BG	5	10	(−9.10e−3, 173)	19,813	21,139	23,251	(−3.67e−6, 1.34e−1, −1220)	19,763	20,627	21,557
BIKE-1	3	BF	8	10	(−5.36e−3, 99.7)	19,867	22,171	25,771	(−9.08e−7, 3.02e−2, −248)	19,853	21,523	23,339
BIKE-1	3	BF	9	9	(−6.14e−3, 114)	19,661	21,661	24,781	(−1.37e−6, 4.71e−2, −403)	19,661	21,059	22,613
BIKE-1	3	BF	10	5	(−6.51e−3, 120)	19,469	21,379	24,371	(−8.64e−7, 2.69e−2, −204)	19,469	21,011	22,787
BIKE-1	3	BF	11	6	(−7.05e−3, 130)	19,373	21,101	23,869	(−1.66e−6, 5.69e−2, −488)	19,373	20,693	22,067

F Illustration Graphs



(a) $r = 9803$, iteration 1

(b) $r = 10163$, iteration 1

(c) $r = 9803$, iteration 2

(d) $r = 10163$, iteration 2

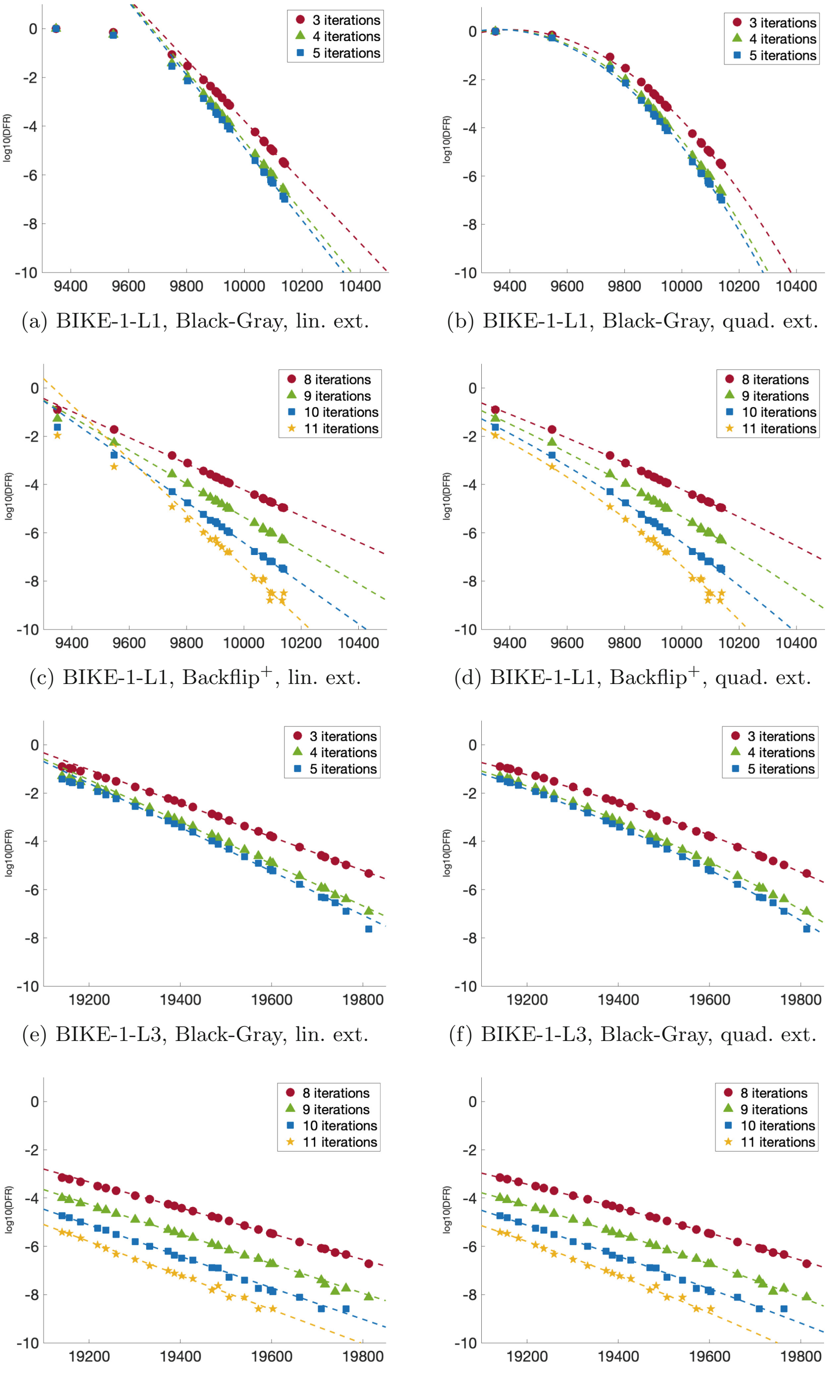
(e) $r = 9803$, iteration 3

(f) $r = 10163$, iteration 3

(g) $r = 9803$, iteration 4

(h) $r = 10163$, iteration 4

Fig. 7. Histograms of the cases (vertical axis; measured in percentage) that end-up with some weight of an "ideal" errors vector (horizontal axis) after the $X_{BG} = 1, 2, 3, 4$ iterations. The decoder is the Black-Gray decoder. Panels a, c, e, g represents the results for $r = 9,803$ and Panels b, d, f, h for $r = 10,163$ with $f = 0, 20, 30, 40$. A lower error weight is better. See explanation in the text.



(a) BIKE-1-L1, Black-Gray, lin. ext.

(b) BIKE-1-L1, Black-Gray, quad. ext.

(c) BIKE-1-L1, Backflip$^+$, lin. ext.

(d) BIKE-1-L1, Backflip$^+$, quad. ext.

(e) BIKE-1-L3, Black-Gray, lin. ext.

(f) BIKE-1-L3, Black-Gray, quad. ext.

(g) BIKE-1-L3, Backflip$^+$, lin. ext.

(h) BIKE-1-L3, Backflip$^+$, quad. ext.

Fig. 8. BIKE-1 Level-1 and Level-3 extrapolations (see Sect. 4 for details).

References

1. C library for quantum-safe cryptography (2019). https://github.com/open-quantum-safe/liboqs/pull/554
2. Melchor, C.A., et al.: Hamming Quasi-Cyclic (HQC) (2017). https://pqc-hqc.org/doc/hqc-specification_2017-11-30.pdf
3. Aragon, N., et al.: BIKE: Bit Flipping Key Encapsulation (2017). https://bikesuite.org/files/round2/spec/BIKE-Spec-2019.06.30.1.pdf
4. Bardet, M., Dragoi, V., Luque, J.G., Otmani, A.: Weak keys for the quasi-cyclic MDPC public key encryption scheme. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2016. LNCS, vol. 9646, pp. 346–367. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31517-1_18
5. Chaulet, J., Sendrier, N.: Worst case QC-MDPC decoder for McEliece cryptosystem. In: 2016 IEEE International Symposium on Information Theory (ISIT), pp. 1366–1370 (2016). https://doi.org/10.1109/ISIT.2016.7541522
6. Drucker, N., Gueron, S.: A toolbox for software optimization of QC-MDPC code-based cryptosystems. J. Cryptogr. Eng. **9**(4), 341–357 (2019). https://doi.org/10.1007/s13389-018-00200-4
7. Drucker, N., Gueron, S.: Additional implementation of BIKE (2019). https://bikesuite.org/additional.html
8. Eaton, E., Lequesne, M., Parent, A., Sendrier, N.: QC-MDPC: a timing attack and a CCA2 KEM. In: Lange, T., Steinwandt, R. (eds.) PQCrypto 2018. LNCS, vol. 10786, pp. 47–76. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-79063-3_3
9. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_34
10. Gallager, R.: Low-density parity-check codes. IRE Trans. Inf. Theory **8**(1), 21–28 (1962). https://doi.org/10.1109/TIT.1962.1057683
11. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 789–815. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_29
12. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10677, pp. 341–371. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_12
13. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. Cryptology ePrint Archive, Report 2017/604 (2017). https://eprint.iacr.org/2017/604
14. Maurich, I.V., Oder, T., Güneysu, T.: Implementing QC-MDPC McEliece encryption. ACM Trans. Embed. Comput. Syst. **14**(3), 44:1–44:27 (2015). https://doi.org/10.1145/2700102. http://doi.acm.org/10.1145/2700102
15. Nilsson, A., Johansson, T., Wagner, P.S.: Error amplification in code-based cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **1**, 238–258 (2019). https://doi.org/10.13154/tches.v2019.i1.238-258
16. NIST: Post-Quantum Cryptography (2019). https://csrc.nist.gov/projects/post-quantum-cryptography. Accessed 20 Aug 2019
17. Gaudry, P., Brent, R.P.Z., Thome, E.: gf2x-1.2 (2017). https://gforge.inria.fr/projects/gf2x/

18. Samardjiska, S., Santini, P., Persichetti, E., Banegas, G.: A reaction attack against cryptosystems based on LRPC codes. In: Schwabe, P., Thériault, N. (eds.) LATINCRYPT 2019. LNCS, vol. 11774, pp. 197–216. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30530-7_10

19. Santini, P., Battaglioni, M., Chiaraluce, F., Baldi, M.: Analysis of reaction and timing attacks against cryptosystems based on sparse parity-check codes. In: Baldi, M., Persichetti, E., Santini, P. (eds.) CBC 2019. LNCS, vol. 11666, pp. 115–136. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25922-8_7

20. Sendrier, N., Vasseur, V.: On the decoding failure rate of QC-MDPC bit-flipping decoders. In: Ding, J., Steinwandt, R. (eds.) PQCrypto 2019. LNCS, vol. 11505, pp. 404–416. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25510-7_22

21. Shoup, V.: Number theory C++ library (NTL) version 11.3.4 (2019). http://www.shoup.net/ntl

22. Wafo-Tapa, G., Bettaieb, S., Bidoux, L., Gaborit, P.: A practicable timing attack against HQC and its countermeasure. Technical report 2019/909 (2019). https://eprint.iacr.org/2019/909