



**HAL**  
open science

## **BIKE: Bit Flipping Key Encapsulation**

Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy,  
Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron,  
Tim Güneysu, et al.

► **To cite this version:**

Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, et al.. BIKE: Bit Flipping Key Encapsulation: Round 4 Submission. 2022. hal-04278509

**HAL Id: hal-04278509**

**<https://inria.hal.science/hal-04278509>**

Submitted on 10 Nov 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Public Domain

# BIKE: Bit Flipping Key Encapsulation



Round 4 Submission

Nicolas Aragon, University of Limoges, France

Paulo S. L. M. Barreto, University of Washington Tacoma, USA

Slim Bettaieb, Worldline, France

Loïc Bidoux, Worldline, France

Olivier Blazy, University of Limoges, France

Jean-Christophe Deneuville, ENAC, Federal University of Toulouse, France

Philippe Gaborit, University of Limoges, France

Santosh Ghosh, Intel, USA

Shay Gueron, University of Haifa, and Amazon Web Services, Israel

Tim Güneysu, Ruhr-Universität Bochum, and DFKI, Germany,

Carlos Aguilar Melchor, University of Toulouse, France

Rafael Misoczki, Google, USA

Edoardo Persichetti, Florida Atlantic University, USA

Jan Richter-Brockmann, Ruhr-Universität Bochum

Nicolas Sendrier, INRIA, France

Jean-Pierre Tillich, INRIA, France

Valentin Vasseur, INRIA, France

Gilles Zémor, IMB, University of Bordeaux, France

**Submitters:** The team listed above is the principal submitter. There are no auxiliary submitters.

**Inventors/Developers:** Same as the principal submitter. Relevant prior work is credited where appropriate.

**Implementation Owners:**

- Reference Software Implementation: Submitters, Amazon Web Services, Intel Corporation, Worldline, Google.
- Additional Software Implementation: Nir Drucker, Shay Gueron, and Dusan Kostic.
- Hardware Implementation: Jan Richter-Brockmann, Ming-Shing Chen, Santosh Ghosh and Tim Güneysu.

**Email Address (preferred):** rafaelmisoczki@google.com

**Postal Address (if absolutely necessary):**

Rafael Misoczki, Google, 1600 Amphitheatre Pkwy, Mountain View CA, 94043, USA.

**Signature:** x. See also printed version of "Statement by Each Submitter".

**Version:** 5.1

**Release Date:** 10/10/2022

## Changes in BIKE for Round 4

- **Data-oblivious sampling technique:** We have changed our approach for sampling fixed-weight vectors to a data oblivious technique, which eliminates the risk of certain side-channel attacks. This change does not lead to any noticeable performance impact given the negligible contribution this step has to the overall BIKE algorithms. New KATs have been generated.

## Changes in BIKE for Round 3

- **Single Variant:** We have narrowed down the set of BIKE variants to a single variant. The single variant is built from the old BIKE-2 with the algorithmic flow adjusted to match the state-of-the-art semantically secure transform. The parameters are chosen to target IND-CCA security.
- **Spec Simplification:** We have made a significant effort to simplify our specification document. Among other things, we have refactored the document structure and moved most of the mathematical background to the Appendix. The body of the document is now more focused on describing the core BIKE techniques rather than recalling the required mathematical background. The overall document continues to be self-content though.
- **Recommended Decoder:** The recommended decoder was changed to Black-Gray-Flip (BGF). It features a secure and efficient fixed-number-of-steps definition, and enjoys a more refined DFR estimate aiming at IND-CCA security. See Section 2.3.
- **Decoding Failure Rate (DFR):** We have extended the DFR discussion to clarify what is the state-of-art on this topic. See Section 3.4.
- **Parameter for NIST Security Level 5:** In response to NIST request after Round 2 selection, we have provided new BIKE parameters targeting security level 5. See Section 2.6.
- **New Hardware Design:** We have extended our hardware design to implement all key generation, encapsulation and decapsulation procedures. This work represents our fastest VHDL implementation of BIKE. See Section 5.4.
- **Replace ParallelHash by normal hashing:** We have replaced the Parallel-Hash procedure by normal hashing (SHA384), leading to superior performance for our reference implementation. See Section 2.5.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Notation and Definition . . . . .	1
1.2	Public-Key Encryption From QC-MDPC Codes . . . . .	1
<b>2</b>	<b>Specification (2.B.1)</b>	<b>2</b>
2.1	Setup . . . . .	2
2.2	The BIKE Key Encapsulation Mechanism . . . . .	3
2.3	Decoder . . . . .	3
2.4	Pseudorandom Bits Generation . . . . .	5
2.5	The Functions <b>H, K, L</b> . . . . .	7
2.6	BIKE Parameters . . . . .	7
<b>3</b>	<b>The Security of BIKE (2.B.4/2.B.5)</b>	<b>8</b>
3.1	Claims . . . . .	8
3.2	Quantum Adversary . . . . .	8
3.3	Known Attacks . . . . .	9
3.4	More About Low DFR . . . . .	10
3.5	Practical security considerations for using BIKE . . . . .	11
<b>4</b>	<b>Design Rationale and Considerations (2.B.6)</b>	<b>11</b>
4.1	What is BIKE and how should it be used? . . . . .	11
4.1.1	What is BIKE in one sentence? . . . . .	11
4.1.2	How many versions of BIKE are proposed? . . . . .	11
4.1.3	How should BIKE be used? . . . . .	12
4.1.4	Under which condition is BIKE IND-CPA? . . . . .	12
4.1.5	Under which condition is BIKE IND-CCA? . . . . .	12
4.1.6	What happens if a key pair is inadvertently used twice? . . . . .	12
4.2	Interoperability . . . . .	13
4.2.1	Can BIKE be used with a different decoder? . . . . .	13
4.2.2	Does the decoder have to check for a decoding failure? . . . . .	13
4.2.3	Can BIKE be used with another pseudorandom generator? . . . . .	13
4.2.4	Can BIKE be used with a smaller block size ( $r$ )? . . . . .	13
4.3	Design rationale . . . . .	14
4.3.1	How is BIKE constructed? . . . . .	14
4.3.2	What happened to the previous versions of BIKE? . . . . .	14
4.3.3	Is BIKE the same as the previously-known BIKE-2-CCA? . . . . .	14
4.3.4	Why keep the Fujisaki-Okamoto transformation? . . . . .	14

4.3.5	Why is BIKE designed over the Niederreiter framework? . . .	14
4.3.6	How can BIKE support polynomial inversion in KeyGen? . . .	15
4.3.7	How was the block length $r$ chosen? . . . . .	15
4.3.8	How were the functions $\mathbf{H}, \mathbf{K}, \mathbf{L}$ designed? . . . . .	15
4.3.9	How was the pseudorandom generation determined? . . . . .	15
<b>5</b>	<b>BIKE Performance (2.B.2)</b>	<b>16</b>
5.1	Memory and Communication Bandwidth . . . . .	16
5.2	Reference Implementation . . . . .	17
5.3	Additional Software Implementation . . . . .	17
5.4	Hardware Implementation . . . . .	19
<b>6</b>	<b>Known Answer Tests – KAT (2.B.3)</b>	<b>20</b>
6.1	KAT for BIKE . . . . .	20
<b>7</b>	<b>Acknowledgments</b>	<b>20</b>
<b>A</b>	<b>Mathematical Background</b>	<b>26</b>
A.1	QC-MDPC Codes . . . . .	26
A.1.1	Circulant Matrices and Quasi-Cyclic Codes . . . . .	26
A.1.2	Circulant Matrices as a Polynomial Ring . . . . .	26
A.1.3	Definition of QC-MDPC Codes . . . . .	27
A.2	Decoding QC-MDPC Codes . . . . .	28
A.2.1	Decoding Algorithm for QC-MDPC Codes . . . . .	28
A.2.2	Decoding Algorithm for BIKE . . . . .	28
A.2.3	Black-Gray Decoding . . . . .	29
A.2.4	Estimating the DFR for High Block Size . . . . .	30
<b>B</b>	<b>Known Attacks</b>	<b>30</b>
B.1	Hard Computational Problems . . . . .	30
B.1.1	Hard Quasi-Cyclic Computational Problems . . . . .	31
B.2	Information Set Decoding . . . . .	32
B.2.1	Exploiting the Quasi-Cyclic Structure. . . . .	33
B.2.2	Exploiting Quantum Computations. . . . .	33
B.3	Vulnerabilities Due to Decoding Failure . . . . .	35
B.3.1	The GJS Reaction Attack . . . . .	35
B.3.2	Proving the DFR – Weak Keys and Error Floors . . . . .	36

<b>C</b>	<b>A CCA Proof for BIKE</b>	<b>36</b>
C.1	An IND-CPA Proof for BIKE PKE . . . . .	37
C.1.1	From Computational Problems to OW-CPA . . . . .	37
C.1.2	From OW-CPA to IND-CPA . . . . .	38
C.2	From IND-CPA to IND-CCA . . . . .	41
C.2.1	PKE Correction and DFR . . . . .	41
C.2.2	HHK Proof . . . . .	41
C.3	The BIKE Key Encapsulation Mechanism . . . . .	43
C.3.1	Concrete Security and Parameters Selection . . . . .	44
C.4	Security with Non Uniform Key Generation . . . . .	45

# 1 Introduction

Detailed mathematical background, decoder availability and performance, security assumptions, and design rationale are discussed later in the document. This section contains basic material relative to the completeness and the soundness of the specification.

## 1.1 Notation and Definition

NOTATION

---

$\mathbb{F}_2$ :	Binary finite field.
$\mathcal{R}$ :	Cyclic polynomial ring $\mathbb{F}_2[X]/(X^r - 1)$ .
$\mathcal{H}_w$ :	Private key space $\{(h_0, h_1) \in \mathcal{R}^2 \mid  h_0  =  h_1  = w/2\}$
$\mathcal{E}_t$ :	Error space $\{(e_0, e_1) \in \mathcal{R}^2 \mid  e_0  +  e_1  = t\}$
$ g $ :	Hamming weight of a binary polynomial $g \in \mathcal{R}$ .
$u \stackrel{\$}{\leftarrow} U$ :	Variable $u$ is sampled uniformly at random from the set $U$ .
$\oplus$ :	exclusive or of two bits, componentwise with vectors

---

**Parameters.** The block size  $r$  (the code length  $n = 2r$ ), the row weight  $w \approx \sqrt{n}$  ( $w$  even and  $w/2$  odd), and the error weight  $t \approx \sqrt{n}$ .

**QC-MDPC Code.** A Quasi-Cyclic Moderate Density Parity Check code of index 2, length  $n$ , and row weight  $w$  is defined as a pair of sparse parity polynomials  $(h_0, h_1) \in \mathcal{H}_w$ .

**Decoder.** Takes as input a syndrome  $s \in \mathcal{R}$  and parity polynomials  $(h_0, h_1) \in \mathcal{H}_w$  and outputs a sparse vector  $(e_0, e_1) \in \mathcal{R}^2$ . With high probability, the decoder verifies

$$((e_0, e_1) \in \mathcal{R}^2 \text{ and } |e_0| + |e_1| \leq t) \Rightarrow (e_0, e_1) = \text{decoder}(e_0 h_0 + e_1 h_1, h_0, h_1).$$

## 1.2 Public-Key Encryption From QC-MDPC Codes

The McEliece scheme [28] can be instantiated with QC-MDPC codes [29]. It is outlined, using the equivalent Niederreiter scheme [30], in Table 1, where the plaintext is represented by the sparse vector  $(e_0, e_1)$ , and the ciphertext by its syndrome  $s$ . The security of the scheme reduces to quasi-cyclic variants of hard problems from coding theory [6, 1], taking the form of distinguishing problems as given in Table 2.



<b>Private Key:</b> $(h_0, h_1) \in \mathcal{H}_w$	<b>Encryption:</b> $(e_0, e_1) \in \mathcal{E}_t \mapsto s = e_0 + e_1 h \in \mathcal{R}$
<b>Public Key:</b> $h = h_1 h_0^{-1} \in \mathcal{R}$	<b>Decryption:</b> $s \in \mathcal{R} \mapsto \text{decoder}(s h_0, h_0, h_1) \in \mathcal{E}_t$

Table 1: QC-MDPC-McEliece: Niederreiter-like PKE from QC-MDPC codes

<b>Key:</b>	distinguish $h_1 h_0^{-1}$ from random, $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$
<b>Message:</b>	distinguish $(e_0 + e_1 h, h)$ from random, $((e_0, e_1), h) \xleftarrow{\$} \mathcal{E}_t \times \mathcal{R}$

Table 2: Hard problem for the security of QC-MDPC-McEliece

## 2 Specification (2.B.1)

### 2.1 Setup

**Input:** Target security level  $\lambda$ .

**Output:** Parameters  $\{r, w, t, \ell\}$ , hash functions  $\{\mathbf{H}, \mathbf{K}, \mathbf{L}\}$ , and decoder.

1. **System Parameters.** Select  $r, w, t, \ell$  following the guidelines in C.3.1.
  - $r$  (block length): a prime number such that 2 is primitive modulo  $r$ .
  - $w$  (row weight): an even positive integer such that  $w/2$  is odd.
  - $t$  (error weight): a positive integer.
  - $\ell$  (shared secret size): a positive integer.

Define the message space  $\mathcal{M} = \{0, 1\}^\ell$  and the shared key space  $\mathcal{K} = \{0, 1\}^\ell$ .

2. **Hash Functions.** Select the functions  $\mathbf{H}, \mathbf{K}, \mathbf{L}$  uniformly at random from the set of functions with the following respective domains and ranges.
  - $\mathbf{H} : \mathcal{M} \rightarrow \mathcal{E}_t$ .
  - $\mathbf{K} : \mathcal{M} \times \mathcal{R} \times \mathcal{M} \rightarrow \mathcal{K}$ .
  - $\mathbf{L} : \mathcal{R}^2 \rightarrow \mathcal{M}$

The functions are modeled as random oracles. A concrete instantiation of  $\{\mathbf{H}, \mathbf{K}, \mathbf{L}\}$  needs to be associated with the scheme.

3. **Decoder.** Select `decoder`, which takes as input  $s \in \mathcal{R}$  and  $(h_0, h_1) \in \mathcal{H}_w$ . The call `decoder( $s, h_0, h_1$ )` returns either  $(e_0, e_1) \in \mathcal{R}^2$  such that  $e_0 h_0 + e_1 h_1 = s$  or the failure symbol  $\perp$ . The decoding failure rate is defined as

$$\text{DFR}(\text{decoder}) = \Pr[(e_0, e_1) \neq \text{decoder}(e_0 h_0 + e_1 h_1, h_0, h_1)]$$

when  $((h_0, h_1), (e_0, e_1))$  is drawn uniformly from  $\mathcal{H}_w \times \mathcal{E}_t$ .

## 2.2 The BIKE Key Encapsulation Mechanism

<p><b>KeyGen</b> : <math>() \mapsto (h_0, h_1, \sigma), h</math>  Output: <math>(h_0, h_1, \sigma) \in \mathcal{H}_w \times \mathcal{M}, h \in \mathcal{R}</math>  1: <math>(h_0, h_1) \xleftarrow{\mathcal{D}} \mathcal{H}_w</math> <span style="float: right;"><math>\triangleright</math> <sup>(1)</sup></span>  2: <math>h \leftarrow h_1 h_0^{-1}</math>  3: <math>\sigma \xleftarrow{\mathcal{S}} \mathcal{M}</math></p>	<p><b>Encaps</b> : <math>h \mapsto K, c</math>  Input: <math>h \in \mathcal{R}</math>  Output: <math>K \in \mathcal{K}, c \in \mathcal{R} \times \mathcal{M}</math>  1: <math>m \xleftarrow{\mathcal{S}} \mathcal{M}</math>  2: <math>(e_0, e_1) \leftarrow \mathbf{H}(m)</math>  3: <math>c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))</math>  4: <math>K \leftarrow \mathbf{K}(m, c)</math></p>
<p><b>Decaps</b> : <math>(h_0, h_1, \sigma), c \mapsto K</math>  Input: <math>((h_0, h_1), \sigma) \in \mathcal{H}_w \times \mathcal{M}, c = (c_0, c_1) \in \mathcal{R} \times \mathcal{M}</math>  Output: <math>K \in \mathcal{K}</math>  1: <math>e' \leftarrow \text{decoder}(c_0 h_0, h_0, h_1)</math> <span style="float: right;"><math>\triangleright e' \in \mathcal{R}^2 \cup \{\perp\}</math></span>  2: <math>m' \leftarrow c_1 \oplus \mathbf{L}(e')</math> <span style="float: right;"><math>\triangleright</math> with the convention <math>\perp = (0, 0)</math></span>  3: <b>if</b> <math>e' = \mathbf{H}(m')</math> <b>then</b> <math>K \leftarrow \mathbf{K}(m', c)</math> <b>else</b> <math>K \leftarrow \mathbf{K}(\sigma, c)</math></p>	

<sup>(1)</sup>:  $\mathcal{D}$  a distribution stemming from Algorithm 3, the constant weight sampler of §2.4

Table 3: The BIKE Key Encapsulation Mechanism

## 2.3 Decoder

The selected decoder is the Black-Gray-Flip (BGF) defined in [16]. It is specified in Algorithm 1 and takes as inputs a vector  $s \in \mathbb{F}_2^r$  and a matrix  $H \in \mathbb{F}_2^{r \times n}$ . The matrix  $H = (H_0 \mid H_1)$  is built from two circulant blocks  $H_0, H_1$  derived from  $(h_0, h_1) \in \mathcal{H}_w$  (see §A.1). The algorithm is defined for every set of system parameters  $(r, w, t)$  (which also determines  $d = w/2$  and  $n = 2r$ ). It is characterized by three other parameters. The first is NbIter, the number of iterations that it runs. The second is  $\tau$ , a threshold gap used to determined the size of the 'gray' set of positions. The third parameter is the threshold function (see below). Relevant values of NbIter,  $\tau$ , and of the threshold function must be specified for every parameter set. The algorithm invokes two functions specified as follows:

- **ctr**( $H, s, j$ ). This function computes a quantity referred to as the *counter* (aka the *number of unsatisfied parity-checks*) of  $j$ . It is the number of '1' (set bits) that appear in the same position in the syndrome  $s$  and in the  $j$ -th column of the matrix  $H$ .

---

**Algorithm 1** Black-Gray-Flip (BGF)

---

**Parameters:**  $r, w, t, d = w/2, n = 2r$  ; NbIter,  $\tau$ , threshold (see text for details)

---

**Require:**  $s \in \mathbb{F}_2^r, H \in \mathbb{F}_2^{r \times n}$ 

```
1:  $e \leftarrow 0^n$ 
2: for  $i = 1, \dots, \text{NbIter}$  do
3:    $T \leftarrow \text{threshold}(|s + eH^\top|, i)$ 
4:    $e, \text{black}, \text{gray} \leftarrow \text{BFIter}(s + eH^\top, e, T, H)$ 
5:   if  $i = 1$  then
6:      $e \leftarrow \text{BFMaskedIter}(s + eH^\top, e, \text{black}, (d + 1)/2 + 1, H)$ 
7:      $e \leftarrow \text{BFMaskedIter}(s + eH^\top, e, \text{gray}, (d + 1)/2 + 1, H)$ 
8:   if  $s = eH^\top$  then
9:     return  $e$ 
10:  else
11:    return  $\perp$ 

12: procedure  $\text{BFIter}(s, e, T, H)$ 
13: for  $j = 0, \dots, n - 1$  do
14:   if  $\text{ctr}(H, s, j) \geq T$  then
15:      $e_j \leftarrow e_j \oplus 1$ 
16:      $\text{black}_j \leftarrow 1$ 
17:   else if  $\text{ctr}(H, s, j) \geq T - \tau$  then
18:      $\text{gray}_j \leftarrow 1$ 
19: return  $e, \text{black}, \text{gray}$ 

20: procedure  $\text{BFMaskedIter}(s, e, \text{mask}, T, H)$ 
21: for  $j = 0, \dots, n - 1$  do
22:   if  $\text{ctr}(H, s, j) \geq T$  then
23:      $e_j \leftarrow e_j \oplus \text{mask}_j$ 
24: return  $e$ 
```

---

- **threshold**( $S, i$ ). This function is the threshold selection rule. It depends, in general, on the syndrome weight  $S$ , the iteration number  $i$ , and on the system parameters. This function is a parameter of the algorithm, it impacts the decoding performance.

**A note about BGF and IND-CCA security.** The DFR of the BGF decoder has been studied by means of simulations and extrapolations in [16]. These techniques provide a strong indication that the DFR is (sufficiently) small with the recommended parameters. This indication may be acceptable from a practical viewpoint, and could be strengthened by further studies. However, at the moment, the current analysis gives only an estimation of the DFR, and not a proven upper bound. Consequently, the BIKE instantiation with the BGF decoder does not make a formal claim for IND-CCA security, although by any practical considerations, this is probably the case.

## 2.4 Pseudorandom Bits Generation

KeyGen, Encaps, and Decaps involve three types of pseudorandom bits stream generation.

- With no constraints on the output (Algorithm 2).
- With a specific weight  $w$  (Algorithm 3).

**Remark 1.** *Algorithm 3 generates a list of  $w$  distinct positions between 0 and  $r - 1$ . This list is also viewed, interchangeably, as the support of a string  $U$  of  $r$  bits (where  $|U| = w$ ).*

*Contrary to the previous version of BIKE's constant weight sampler, Algorithm 3 never rejects data and is thus immune to the attack [19]. It is shown in [37] that the returned subset is almost uniform among the subset of  $\{0, \dots, r - 1\}$  of cardinality  $w$ , close enough to uniform to have no effective impact on security (for more details we refer the reader to Sections 3 and 4 of [37]).*

**Key Generation.** The secret key  $(h_0, h_1)$  in Table 3 is sampled according to a distribution  $\mathcal{D}$  over  $\mathcal{H}_w$ . To avoid code duplication this distribution can reuse Algorithm 3, used to implement the hash function  $\mathbf{H}$ , with input  $(\text{len}, \text{wt}) = (r, w/2)$ . Two calls are required, for  $h_0$  then  $h_1$ . The corresponding distribution is biased, however the impact of this bias on security is hardly measurable as shown in §C.4.

**SHAKE256 based pseudorandom bits generation.** The building block for these algorithms is SHAKE256. Suppose that `seed` is a 256-bit seed and  $\mu$  is a positive integer. Denote the  $\mu$  blocks (of 1088 bits each) output of SHAKE256 with that seed by  $\text{SHAKE256}(\text{seed}, \mu)$ . Let  $\nu$  be a positive integer. Then, the least significant  $\nu$  bits of  $\text{SHAKE256}(\text{seed}, \text{ceil}(\nu/1344))$  are denoted by  $\text{SHAKE256-Stream}(\text{seed}, \nu)$ .

---

**Algorithm 2**  $\text{GenPseudoRand}(\text{seed}, \text{len})$

---

**Require:** `seed` (32 bytes)

1: **return**  $\text{SHAKE256-Stream}(\text{seed}, \text{len})$

---



---

**Algorithm 3**  $\text{WSHAKE256-PRF}(\text{seed}, \text{len}, \text{wt})$

---

**Require:** `seed` (32 bytes), `len`, `wt`

**Ensure:** A list (`wlist`) of `wt` distinct elements in  $\{0, \dots, \text{len} - 1\}$ .

1: `wlist`  $\leftarrow$  () ▷ empty list

2:  $s_0, \dots, s_{\text{wt}-1} \leftarrow \text{SHAKE256-Stream}(\text{seed}, 32 \cdot \text{wt})$   
▷ parse as a sequence of `wt` non negative 32-bits integers

3: **for**  $i = (\text{wt} - 1), \dots, 1, 0$  **do** ▷  $i$  decreasing from `wt` - 1 to 0

4:     `pos`  $\leftarrow i + \lfloor (\text{len} - i) s_i / 2^{32} \rfloor$

5:     `wlist`  $\leftarrow \text{wlist}, (\text{pos} \in \text{wlist}) ? i : \text{pos}$

6: **return** `wlist`

---

## 2.5 The Functions $\mathbf{H}$ , $\mathbf{K}$ , $\mathbf{L}$

The functions  $\mathbf{H}$ ,  $\mathbf{K}$ ,  $\mathbf{L}$  are modeled as random oracles. Their concrete instantiation is the following.

- $\mathbf{H}$  is instantiated as a pseudorandom expansion of a seed of length  $\ell$  bits that is input to the function. It is generated by invoking Algorithm 3 with the appropriate parameters.
- $\mathbf{K}$  is instantiated as the  $\ell = 256$  least significant bits of the standard SHA384 hash digest of the input. The notation  $\mathbf{K}(m, C)$  where  $C = (c_0, c_1)$  (and similarly,  $\mathbf{K}(m', C)$ ) refers to hashing an input of  $\{0, 1\}^{\ell+r+\ell}$  bits that is the concatenation of  $m$ ,  $c_0$  and  $c_1$ . Here, the bits of  $m$  are consumed (by SHA384) first, then the bits of  $c_0$ , and then the bits of  $c_1$ .
- $\mathbf{L}$  is instantiated as the  $\ell = 256$  least significant bits of the standard SHA384 hash digest of the input. The notation  $\mathbf{L}(e_0, e_1)$  (and similarly,  $\mathbf{L}(e'_0, e'_1)$ ) refers to hashing an input of  $\{0, 1\}^{r+r}$  bits that is the concatenation of  $e_0$  and  $e_1$ . Here, the bits of  $e_0$  are consumed (by SHA384) first, and then then the bits of  $e_1$ .

## 2.6 BIKE Parameters

The NIST call for proposals indicates several security categories that are related to the hardness of a key search attack on a block cipher, like AES. BIKE targets security levels 1, 3, and 5, corresponding to the security of AES-128, AES-192, and AES-256, respectively.

For all security levels, the key length parameter is fixed to  $\ell = 256$ . A parameter set for BIKE is a triple  $(r, w, t)$ . The suggested parameters are summarized in Table 4.

Security	$r$	$w$	$t$	DFR <sup>†</sup>
Level 1	12,323	142	134	$2^{-128}$
Level 3	24,659	206	199	$2^{-192}$
Level 5	40,973	274	264	$2^{-256}$

Table 4: Suggested BIKE Parameters.

<sup>†</sup> The DFR in Table 4 is estimated for the BGF decoder of §2.3 with the following additional parameters (note that here, `threshold` is independent of the iteration number  $i$ ):

- For Level 1:  $\text{NbIter} = 5$ ,  $\tau = 3$ ,  
 $\text{threshold}(S, i) = \max(\lfloor 0.0069722 \cdot S + 13.530 \rfloor, 36)$
- For Level 3:  $\text{NbIter} = 5$ ,  $\tau = 3$ ,  
 $\text{threshold}(S, i) = \max(\lfloor 0.005265 \cdot S + 15.2588 \rfloor, 52)$
- For Level 5:  $\text{NbIter} = 5$ ,  $\tau = 3$ ,  
 $\text{threshold}(S, i) = \max(\lfloor 0.00402312 \cdot S + 17.8785 \rfloor, 69)$

### 3 The Security of BIKE (2.B.4/2.B.5)

This section discusses various security aspects relative to BIKE. It is assumed here that the instantiation of  $\mathbf{H}, \mathbf{K}, \mathbf{L}$  is an acceptable approximation for random oracles.

#### 3.1 Claims

- BIKE is proven IND-CPA secure under assumptions 1 and 2
- BIKE is proven IND-CCA secure under assumptions 1, 2, and 3

**Assumption 1.** Hardness of  $\text{QCSD}_{r,t}$

**Assumption 2.** Hardness of  $\text{QCCF}_{r,w}$

**Assumption 3.** Correctness of decoder

where  $r, w, t$  and `decoder` are parameters defined in the system setup §2.

The first two assumptions relate to standard hard problems from coding theory, respectively decoding and codeword finding in an arbitrary quasi-cyclic code, see §B.1.1.

Correctness in the third assumption refers to [23] where a KEM is  $\delta$ -correct if the decapsulation fails (*i.e.* disagrees with encapsulation) with probability at most  $\delta$  on average over all keys and messages. Similarly, a decoder will be  $\delta$ -correct if its failure rate is at most  $\delta$  on average when the input is drawn uniformly. This matches the DFR definition in the setup §2.

#### 3.2 Quantum Adversary

The proof framework of [23] is also valid in the QROM. Tightness was later improved in [35]. In view of those works, and following [24] and Proposition 1 of §B.2.2, it appeared safe to use  $\lambda = 128, 192, 256$  bits of classical security to meet respectively levels 1, 3, and 5 of NIST target security.

### 3.3 Known Attacks

Known attacks are detailed in appendix §B. The first two assumptions relate to hard computational problems from coding theory. Parameters are selected so that the workfactor of best known solvers for those problems, variants of Information Set Decoding, are above the required security level. See §B.2.

The third assumption relates to the average decoding failure rate (DFR), as defined in §2, of the chosen decoder. The reaction attack [20] will allow the secret key recovery if an attacker is able to discover a few decoding failures for the same key. Specifically, if the DFR is  $2^{-S}$  then an attacker may recover the secret after an average computational effort of order  $2^S$ . See §B.3.

It can be observed that the formal security in the classical (*i.e.* non quantum) setting, stated in Theorem 3, relates tightly to the practical security stemming from the above attacks. If the parameters are chosen to resist to the decoding attacks and to the reaction attack, they also match the requirements for formal security.

**Parameter Selection:** To reach  $\lambda$  bits of (classical) IND-CCA security, the parameters are  $r, w, t, \ell$ , and `decoder` chosen in the setup such that:

1.  $\text{QCCF}_{r,w}$  offers  $\lambda$  bits of security
2.  $\text{QCSD}_{r,t}$  offers  $\lambda$  bits of security
3.  $|\mathcal{M}| = 2^\ell \geq 2^\lambda$
4.  $\text{DFR}(\text{decoder}) \leq 2^{-\lambda}$ .

The parameters are chosen in the following order:

- Choosing  $\ell \geq \lambda$  is straightforward, in practice  $\ell = 256$  for all parameter sets.
- The computational problems guide the selection of  $w$  and  $t$  based on the best known solvers, as discussed in §B.2. The block size  $r$  has a very limited influence on those solvers' complexity, see §A.2.4.
- Finally, with  $w$  and  $t$  fixed, the block length  $r$  is selected by simulation and extrapolation so that the DFR estimate is low enough, as discussed in §A.2.4.

There are additional requirements for the parameters selection (see §A.1.2): 1) the block size is chosen such that 2 is primitive mod  $r$  to avoid any undesirable structure in the polynomial ring  $\mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$ , and 2) the row weight  $w$  is chosen even and such that  $|h_0| = |h_1| = w/2$  is odd to ensure that  $h_0$  is always invertible in  $\mathcal{R}$ .



### 3.4 More About Low DFR

The failure assumption relates to the decoder (family) used in the system specification. An important fact which derives from the security reduction is that the assumption 3 above captures all IND-CCA issues related to this particular decoder. It is known from [20] that a high DFR leads to a key attack, and from the reduction that a low DFR is enough to resist, not only to the previously mentioned attack, but to any attack that would not also solve a hard code-based computational problem.

The question is now, *what does it take to prove a low DFR?* The current state-of-art looks more like an estimation: simulation data, with fixed  $(w, t)$  and varying block size  $r$ , is extrapolated to determine an upper bound for a suitably secure block size. This extrapolation is consistent with known results, asymptotic [43] and Markovian model of bit flipping [39]. Those results predict the typical behaviour of the decoder, but *may* not take into account specific structures that could hinder decoding. Those are of two kinds, *weak keys*, decoding failures which are caused by structured keys, and *error floors*, decoding failures which are caused by structured errors. Note that in both case the question is not about existence, those objects do exist. The question is whether their contribution to the average DFR dominates, defeating the analysis which only consider the typical case.

**Weak Keys.** Some weak keys were exhibited in [14], they have a low density, but have a strong impact on decoding. Those weak keys were generalized and studied further [41]. However, there are strong indications that they have a negligible impact on the average DFR [45].

**Error Floors.** Low density parity check codes decoding failure rate suffer from a phenomenon known as error floor: when the error rate decreases, the waterfall shape curve of the DFR logarithm eventually turns into a plateau. This error floor effect is due to codewords and near-codewords. A near-codeword is a word of low Hamming weight whose syndrome also has a low Hamming weight. Error patterns which are close to a codeword or a near-codeword are prone to decoding failure. In [40] the effect of codewords on error floors is shown to be negligible. There exists near-codewords for QC-MDPC. For instance, in the polynomial setting, see §A.1.2, relatively to the QC-MDPC parity check matrix  $(h_0, h_1)$ , the syndrome of the word  $(h_0^T, 0)$  is  $h_0^2$ , and both the word and syndrome have weight  $w/2$ . Those near-codewords and their impact on decoding failure have been studied [45, 3], but at this moment it seems difficult to conclude whether or not the contribution of error patterns close to identified near-codewords is dominant in the error floor region of

the DFR, and the low DFR assumption remains partly heuristic.

### 3.5 Practical security considerations for using BIKE

- An instantiation of BIKE can use different decoders without affecting interoperability. The decoder must be implementable in constant-time to avoid side-channel attacks, and its DFR must be low enough to match the security requirement.
- BIKE’s design fits well with ephemeral keys. The party that initiates a session needs to: a) Generate a fresh private/public key pair for every session; b) Refuse to decapsulate more than one incoming ciphertext (presumably the result of a legitimate encapsulation) with that key. The IND-CPA security property suffices for this type of usage.
- BIKE can also support static keys, *i.e.* a long-term use of a single key. This usage requires the IND-CCA security property and therefore a low enough DFR for the specified decoder. However, this usage model implies the loss of forward secrecy.

## 4 Design Rationale and Considerations (2.B.6)

This section explains briefly the design rationale and some considerations about the specification of BIKE, by answering a sequence of questions that may occur.

### 4.1 What is BIKE and how should it be used?

#### 4.1.1 What is BIKE in one sentence?

BIKE is a Key Encapsulation Mechanism (KEM) based on Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) codes, that is proposed for the Post-Quantum Cryptography (PQC) Standardization project of the National Institute of Standards and Technology (NIST).

#### 4.1.2 How many versions of BIKE are proposed?

There is only one version of BIKE, defined with three parameter sets  $(r, w, t)$ : one for Security Level 1, one for Level 3 and one for Level 5. Some additional parameters are associated with the specific BGF decoder that is associated with the proposal.

### 4.1.3 How should BIKE be used?

BIKE is primarily designed to be used in synchronous communication protocols (*e.g.* TLS) with ephemeral keys, *i.e.* with a fresh public/private key pair for every key exchange session. In particular, decapsulation with a given private key should be allowed only once. Such usage model provides forward secrecy. A KEM with IND-CPA security is sufficient for such usage.

Key reuse or adapting BIKE to asynchronous protocols (*e.g.* email) require to secure long term static keys. Those usage models are possible but no longer provides forward secrecy and require IND-CCA security. Note that they are not compliant with BIKE's current specification.

### 4.1.4 Under which condition is BIKE IND-CPA?

IND-CPA security is guaranteed if the parameters are chosen so that the underlying generic quasi-cyclic code-based computational problems are hard enough. Those problems have been studied for a while and are notoriously hard. Best known solvers are well understood and easy to analyze. Proposed parameters for BIKE tightly match those analyses.

### 4.1.5 Under which condition is BIKE IND-CCA?

To reach IND-CCA security, BIKE must be instantiated with a decoder that has a Decoding Failure Rate (DFR) of the required magnitude, see §C and [17]. The BGF decoder that is associated with BIKE targets a DFR of  $2^{-128}$ ,  $2^{-192}$  and  $2^{-256}$  for the respective levels of security. This is indeed the *estimated* DFR. The estimation suggests a high confidence level through a methodology that uses extensive simulations and extrapolations models stemming from asymptotic analyses [43, 39].

### 4.1.6 What happens if a key pair is inadvertently used twice?

Formally, IND-CCA security, that is a low enough DFR, is required to offer a guaranty in case of key reuse. In that case the key may be reused indefinitely. Else, existing attacks [20] require multiple decoding failures to succeed, and even with amplification techniques [31] there is no plausible scenario in which a few reuse of the same key create an effective threat.

## 4.2 Interoperability

### 4.2.1 Can BIKE be used with a different decoder?

Yes, but caution is needed. First, a protocol that uses BIKE should use ephemeral keys. The choice of a different decoder (or the same decoder with different parameters) does not affect interoperability. Such a choice could potentially speed up Decaps at the expense of increasing the (failure) probability that a session does not end up with a successfully exchanged shared key. As long as this probability is deemed tolerable in the overall system context, applications are free to select a decoder as an implementation choice. This means that decoders can be defined, tuned, and optimized for specific platforms with specific constraints. If an instantiation of BIKE targets IND-CCA security, it must choose a decoder with a (proven) sufficiently low DFR. If BIKE is selected for standardization, NIST could specify a list of allowable decoders, or requirements for allowable decoders.

### 4.2.2 Does the decoder have to check for a decoding failure?

There are equivalent ways to check the set of logical conditions. A decoder `decoder` can be defined to always return an error vector  $(e'_0, e'_1)$  and no other indication. Then, `decoder` succeeded if and only if  $e'_0 h_0 + e'_1 h_1 = s$  and  $|(e'_0, e'_1)| = t$ , and otherwise it failed. Checking these conditions is moved outside the scope of `decoder`, and becomes part of Decaps.

### 4.2.3 Can BIKE be used with another pseudorandom generator?

Yes, but some caution is needed. An alternative pseudorandom generation algorithm can be acceptable if it meets the security requirements (indistinguishability from random strings). An acceptable alternative does not affect interoperability.

If BIKE is selected for standardization, NIST could specify a list of allowable pseudorandom generation algorithms, or requirements for allowable algorithms should be specified.

### 4.2.4 Can BIKE be used with a smaller block size ( $r$ )?

In theory, yes: this could have been specified as an option, but a value of  $r$  affects interoperability. For the sake of simplicity, BIKE is specified with one choice only. The rationale behind this choice is explained in §4.3.7.

## 4.3 Design rationale

### 4.3.1 How is BIKE constructed?

BIKE is built upon the Niederreiter framework, with some tweaks. It also applies the implicit-rejection version of Fujisaki-Okamoto transformation ( $FO^\perp$ , as described in [23]) for converting a  $\delta$ -correct PKE into an IND-CCA KEM.

### 4.3.2 What happened to the previous versions of BIKE?

The previous iteration of the proposal included six variants, namely BIKE-1, BIKE-2, BIKE-3, BIKE-1-CCA, BIKE-2-CCA and BIKE-3-CCA. Following NIST's suggestion to reduce the number of options in the proposal, the designers of BIKE decided to consolidate BIKE to one version only, namely BIKE-2-CCA. It is now called simply BIKE. The previous versions remain available on the website<sup>1</sup>.

### 4.3.3 Is BIKE the same as the previously-known BIKE-2-CCA?

Not exactly, the round 2 specification (v3) was modified in v3.1 in an attempt to conform with the  $FO^\perp$  construction of [23]. Further modifications, the shared secret derivation and the domain separation of the hash functions  $\mathbf{H}, \mathbf{K}, \mathbf{L}$ , were made in [17] to match precisely the  $FO^\perp$  construction of [23] and obtain the IND-CCA security proof.

### 4.3.4 Why keep the Fujisaki-Okamoto transformation?

This is a design choice that targets simplicity. Indeed, it is possible to build a version of BIKE that does not apply the  $FO^\perp$  transformation and targets only IND-CPA security. However, the difference in the performance is negligible (see [14]) and does not justify the complication of maintaining such a design as a separate version.

### 4.3.5 Why is BIKE designed over the Niederreiter framework?

The design of BIKE is based on the Niederreiter framework because it requires only half the communication bandwidth compared to an analogous design over the McEliece framework. The trade-off associated with this choice is the cost of the (polynomial) inversion required for the key generation.

---

<sup>1</sup><https://bikesuite.org>

### 4.3.6 How can BIKE support polynomial inversion in KeyGen?

The cost of polynomial inversion was considered too prohibitive until recently (especially with ephemeral keys usage), but the fast polynomial inversion algorithm proposed in [15] changed the picture. This algorithm is similar to the Itoh-Tsuji inversion algorithm, where the essence is that computing  $a^{2^k}$  is efficient. The Itoh-Tsuji algorithm inverts an element of  $\mathbb{F}_{2^k}$ , where the field elements are represented in normal basis. The new algorithm generalizes it to the ring of polynomials used in BIKE (and other QC-MDPC schemes):  $\mathbb{F}_2[x]/\langle(x-1)h\rangle$  with irreducible  $h$ . Details are provided in [15]. This algorithm is implemented in constant-time and used in the Additional Software Implementation Code Package (see Section 5.3).

### 4.3.7 How was the block length $r$ chosen?

The block length  $r$  determines the sizes of the public key, the ciphertext, and significantly affects the overall latency and the communication bandwidth. By the design of BIKE,  $r$  needs to be prime and satisfy the requirement that  $(X^r - 1)/(X - 1) \in \mathbb{F}_2[X]$  is irreducible. It needs to be sufficiently large to satisfy (together with the choice of  $w$  and  $t$ ) the scheme's security target and the DFR target for the decoder. In addition, [15] suggests that the inversion algorithm is especially efficient if the Hamming weight of  $(r - 2)$ , is small. Indeed, for  $r = 12323$ ,  $|(r - 2)| = 4$ , for  $r = 24659$   $|(r - 2)| = 5$ , and for  $r = 40973$ ,  $|(r - 2)| = 5$ .

### 4.3.8 How were the functions **H**, **K**, **L** designed?

BIKE specification models **H**, **K**, **L** as random oracles. The concrete realization of **K** and **L** relies on the standard SHA3-384 hash function that has sufficient capacity in its compression function, and is accepted by NIST for this purpose. The function **H** uses 256 bits as a key, and SHAKE256 based pseudorandom expansion.

### 4.3.9 How was the pseudorandom generation determined?

The pseudorandom generation uses a SHAKE256 implementation with a 256-bit seed. Therefore, BIKE only relies on one single symmetric cryptographic primitive, i.e., the sponge-construction of KECCAK. This design choice is especially beneficial for hardware devices to reduce the overall footprint by instantiating only one symmetric primitive.

## 5 BIKE Performance (2.B.2)

This section discusses the essential characteristics and performance of BIKE.

### 5.1 Memory and Communication Bandwidth

Table 5 summarizes the minimum memory requirements for BIKE.

Quantity	Size	Level 1	Level 3	Level 5
Private key	$\ell + w \cdot \lceil \log_2(r) \rceil$	2,244	3,346	4,640
Public key	$r$	12,323	24,659	40,973
Ciphertext	$r + \ell$	12,579	24,915	41,229

Table 5: Private Key, Public Key and Ciphertext sizes (in bits).

**Remark 2.** The private key consists of the vectors  $(h_0, h_1) \in \mathcal{R}$  with  $|h_0| = |h_1| = w/2$  and  $(\sigma)$ . Both  $h_0$  and  $h_1$  can be represented by  $r$  bits. Alternatively, a more compact representation is listing the  $w/2$  positions of the set bits. This listing yields a  $(\frac{w}{2} \cdot \lceil \log_2(r) \rceil)$ -bits representation. Therefore, the size for this part of the private key is  $(w \cdot \lceil \log_2(r) \rceil)$ -bits. Since  $\lceil \log_2(r) \rceil < 16$  for the proposed parameter sets, an implementation may prefer (for simplicity) to store these vectors as a sequence of  $w$  16-bits elements. The second part of the private key,  $(\sigma)$ , requires  $\ell$  bits of storage. In total, BIKE private keys can be stored in a container of  $(\ell + w \cdot \lceil \log_2(r) \rceil)$  bits. Applications may choose to explore the possibility of generating the private key on the fly, from a (secured) seed to obtain a favorable memory vs. latency trade-off.

Table 6 shows the communication bandwidth cost per message.

Message Flow	Message	Size	Level 1	Level 3	Level 5
Init. $\rightarrow$ Resp.	$h$	$r$	12,323	24,659	40,973
Resp. $\rightarrow$ Init.	$C$	$r + \ell$	12,579	24,915	41,229

Table 6: BIKE communication bandwidth (in bits).

## 5.2 Reference Implementation

The reference implementation of BIKE is available on BIKE’s official website<sup>2</sup>. It is a pure C implementation intended to provide readability and help researchers get familiarized with the BIKE algorithms. It is not designed to run in constant-time, as required for real-world implementation to offer side-channel resistance. For real-world performance characterization, the reader is referred to the Additional Implementation numbers described in §5.3, which is side-channel protected and leverages efficient platform instruction sets.

## 5.3 Additional Software Implementation

The Additional Software Implementation Code Package for BIKE was developed by Nir Drucker, Shay Gueron, and Dusan Kostic. It is maintained in a github repository<sup>3</sup>. The package includes implementations for several CPU architectures. In particular, it can be compiled for a 64-bit ARM and for x86 processors. ARM architectures are supported by a fully portable implementation written in C. When the code is compiled for x86 processors, the resulting binary contains the following implementations:

- Fully portable.
- Optimized for AVX2 and AVX512 instruction sets.
- Optimized for CPUs that support PCLMULQDQ, and the latest Intel vector-PCLMULQDQ instruction.

When the package is used on an x86 CPU, it automatically (in runtime) detects the CPU capabilities and runs the fastest available code path, based on the detected capabilities. The fully portable version, which is built by default, requires OpenSSL. The library can also be compiled in a "stand-alone" mode, without OpenSSL, but only for a processor that supports AES-NI and AVX instructions.

The package includes testing and it uses the KAT generation utilities provided by NIST. The code is “stand-alone”, i.e., it does not depend on external libraries. All the functionalities available in the package are implemented in *constant-time*, which means that: a) No branch depends on a secret piece of information; b) All the memory access patters are independent of secret information.

---

<sup>2</sup><https://bikesuite.org/reference.html>

<sup>3</sup><https://github.com/aws-labs/bike-kem>



**Performance benchmarking details.** The performance is reported here in processor cycles, and reflects the performance per *single core*. The measurements methodology follows the description in [13].

**The benchmarking platform.** The platforms used in the experiments were equipped with Intel® Xeon® Platinum 8175M CPU @ 2.50GHz, one with AVX512 and `vector-PCLMULQDQ` instruction support, and another with AVX512 support only. The Intel® Turbo Boost Technology was turned off for the experiments in order to force a fixed frequency and consistently measure performance in processor cycles.

**OS and compilation.** The code was compiled with clang (version 10.0.0) in 64-bit mode and ran on a Linux OS (Ubuntu 20.04).

## Performance numbers

Table 7: BIKE Level-1,  $r = 12323$ ,  $w = 142$ ,  $t = 134$ . Decoder BGF with 5 iterations. Performance in  $10^3$  cycles.

	AVX512	Vector-PCLMULQDQ
KeyGen	589	366
Encaps	97	74
Decaps	1,135	1,177

Table 8: BIKE Level-3,  $r = 24659$ ,  $w = 206$ ,  $t = 199$ . Decoder BGF with 5 iterations. Performance in  $10^3$  cycles.

	AVX512	Vector-PCLMULQDQ
KeyGen	1,823	1,049
Encaps	223	164
Decaps	3,887	3,512

**Remark 3.** *A meaningful measure for the efficiency of the KEM, in the case where it is used with ephemeral keys is the cumulative latency of KeyGen and Decaps. The reason is that the communicating party that initiates the exchange executes KeyGen subsequently executes Decaps. The numbers reported in Tables 7 and 8 indicate that KeyGen is significantly faster than Decaps on modern platforms with AVX512 support, and even more so with the newer vector-PCLMULQDQ instruction.*

Table 9: Implementation results of the united BIKE hardware core for Level 1 ( $r = 12323$ ).

Design	Utilization					Performance						
	Logic		Memory		Area	Frequency	Key Gen		Encaps		Decaps	
	LUT	DSP	FF	BRAM	Slices	MHz	cycles <sup>†</sup>	$\mu s$	cycles <sup>†</sup>	$\mu s$	cycles <sup>†</sup>	$\mu s$
Light weight	12319	7	3896	9	3777	121	463	3797	54	443	841	6896
Trade-off	19607	9	5008	17	5617	100	187	1870	28	280	421	4210
High speed	25549	13	5462	34	7332	113	190	1672	15	132	215	1892

<sup>pke</sup> Results are only for the PKE and not for the KEM.

<sup>†</sup> in thousand.

## 5.4 Hardware Implementation

The Hardware Implementation Code for BIKE was developed by Jan Richter-Brockmann, Ming-Shing Chen, Santosh Ghosh and Tim Güneysu and has been presented in [33]. Compared to the first hardware design of BIKE [34], the latest hardware implementation uses a more efficient multiplier and faster inversion module. Additionally, the revised design implements a united hardware core allowing to perform the key generation, encapsulation, and decapsulation. All random oracles are already based on KECCAK.

All the hardware files are published on the BIKE website<sup>4</sup>.

**Implementation Results** The implementation results are summarized in Table 9 including hardware utilization and timing behavior. All results were generated for an Artix-7 FPGA (xc7a200). The exponentiation required for the key generation is accomplished by an algorithm which is based on the extended Euclidean algorithm [8]. The decapsulation is performed by a hardware implementation of the BGF decoder. By instantiating multiple UPC equation counter in parallel, the decoding can be highly accelerated. More details can be found in [34, 33]. Note that the implementations of the random oracles are not updated to prevent the latest timing attacks presented in [19] yet.

<sup>4</sup><https://bikesuite.org/>

## 6 Known Answer Tests – KAT (2.B.3)

### 6.1 KAT for BIKE

The KAT files of BIKE are available in:

- req file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level11_3114.req`
- rsp file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level11_3114.rsp`
- req file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level13_6198.req`
- rsp file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level13_6198.rsp`

## 7 Acknowledgments

Special thanks are extended to Nir Drucker and Dusan Kostic for significant contributions that helped shaping BIKE into its current form. They co-authored the following papers [14] (accepted to CBCrypto 2020), [16] (accepted to PQCrypto 2020), [15] (accepted to CSCML 2020), [17]. They are also co-developers of the Additional Software Implementation Code Package of BIKE (residing in the git repository [github.com/aws-labs/bike-kem](https://github.com/aws-labs/bike-kem)).

Special thanks are extended to Nir Drucker, Shay Gueron and Dusan Kostic for their contribution in reporting an inconsistency in the original Round 4 submission (spec v5.0) submitted to NIST on 10/04/2022. Spec v5.0 did not specify with clarity what sampling algorithm should be used in *KeyGen*, while the reference implementation and KATs arbitrarily reused Algorithm 3 originally designed for *Encaps* and *Decaps*. This led to the team to revisit this design decision, and conclude that Algorithm 3 is suitable for *KeyGen* as well (see Appendix C.4). Spec v5.1, released on 10/10/2022, can therefore be implemented with a single sampling technique (Algorithm 3) instead of two.

Shay Gueron, Tim Güneysu, Nicolas Sendrier and Jean-Pierre Tillich were supported in part by the Commission of the European Communities through the Horizon 2020 program under project number 645622 (PQCRYPTO).

Paulo S. L. M. Barreto was also partially supported by Intel and FAPESP through the project “Efficient Post-Quantum Cryptography for Building Advanced Security

Applications” (grant No. 2015/50520-6).

Shay Gueron was also partially supported by: NSF-BSF Grant 2018640; The Israel Science Foundation (grant No. 3380/19); 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 Bureau in the Prime Minister’s Office.

Tim Güneysu was partially supported by Intel (<http://www.icri-cars.org>).

Edoardo Persichetti was partially supported by NSF Grant CNS 1906360.

## References

- [1] Michael Alekhnovich. More on average case vs approximation complexity. In *FOCS 2003*, pages 298–307. IEEE, 2003.
- [2] Benny Applebaum, Benny Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In *STOC 2010*, pages 171–180, 2010.
- [3] Sarah Arpin, Tyler Raven Billingsley, Daniel Rayor Hast, Jun Bo Lau, Ray A. Perlner, and Angela Robinson. A study of error floor behavior in QC-MDPC codes. In Jung Hee Cheon and Thomas Johansson, editors, *PQCrypto 2022*, volume 13512, pages 89–103. Springer, 2022.
- [4] Marco Baldi, Paolo Santini, and Franco Chiaraluce. Soft mceliece: MDPC code-based mceliece cryptosystems with very compact keys through real-valued intentional errors. In *International Symposium on Information Theory – ISIT’2016*, pages 795–799. IEEE Press, 2016.
- [5] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in  $2^{n/20}$ : How  $1+1=0$  improves information set decoding. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, 2012.
- [6] Elwyn Berlekamp, Robert J. McEliece, and Henk van Tilborg. On the inherent intractability of certain coding problems (corresp.). *Information Theory, IEEE Transactions on*, 24(3):384 – 386, may 1978.

- [7] Daniel J Bernstein. Grover vs. McEliece. In *International Workshop on Post-Quantum Cryptography*, pages 73–80. Springer, 2010.
- [8] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):340–398, 2019.
- [9] Julia Chaulet and Nicolas Sendrier. Worst case QC-MDPC decoder for McEliece cryptosystem. In *Information Theory (ISIT), 2016 IEEE International Symposium on*, pages 1366–1370. IEEE, 2016.
- [10] Tung Chou. Qcbits: Constant-time small-key code-based cryptography. In Benedikt Gierlich and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 280–300. Springer, 2016.
- [11] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, January 2004.
- [12] Alexander W. Dent. A designer’s guide to kems. In Kenneth G. Paterson, editor, *Cryptography and Coding 2003*, volume 2898 of *LNCS*, pages 133–151. Springer, 2003.
- [13] Nir Drucker and Shay Gueron. A toolbox for software optimization of QC-MDPC code-based cryptosystems. *Journal of Cryptographic Engineering*, pages 1–17, Jan. 2019.
- [14] Nir Drucker, Shay Gueron, and Dusan Kostic. On constant-time QC-MDPC decoding with negligible failure rate. Cryptology ePrint Archive, Report 2019/1289, Nov 2019.
- [15] Nir Drucker, Shay Gueron, and Dusan Kostic. Fast polynomial inversion for post quantum QC-MDPC cryptography. Cryptology ePrint Archive, Report 2020/298, 2020. <https://eprint.iacr.org/2020/298>.
- [16] Nir Drucker, Shay Gueron, and Dusan Kostic. QC-MDPC decoders with several shades of gray. In Jintai Ding and Jean-Pierre Tillich, editors, *PQCrypto 2020*, volume 12100 of *LNCS*, pages 35–50. Springer, 2020.
- [17] Nir Drucker, Shay Gueron, Dusan Kostic, and Edoardo Persichetti. On the applicability of the Fujisaki-Okamoto transformation to the BIKE KEM. *IACR Cryptology ePrint Archive*, 2020.

- [18] R. G. Gallager. *Low-Density Parity-Check Codes*. PhD thesis, M.I.T., 1963.
- [19] 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 Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):223–263, 2022.
- [20] Qian Guo, Thomas Johansson, and Paul Stankovski. *A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors*, pages 789–815. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.
- [21] Yann Hamdaoui and Nicolas Sendrier. A non asymptotic analysis of information set decoding. Cryptology ePrint Archive, Report 2013/162, 2013. <http://eprint.iacr.org/2013/162>.
- [22] Stefan Heyse, Ingo Von Maurich, and Tim Güneysu. Smaller keys for code-based cryptography: QC-MDPC McEliece implementations on embedded devices. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 273–292. Springer, 2013.
- [23] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2017.
- [24] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In Tanja Lange and Tsuyoshi Takagi, editors, *PQCrypto 2017*, volume 10346 of *LNCS*, pages 69–89. Springer, 2017.
- [25] Carl Löndahl, Thomas Johansson, Masoumeh Koochak Shooshtari, Mahmoud Ahmadian-Attari, and Mohammad Reza Aref. Squaring attacks on McEliece public-key cryptosystems using quasi-cyclic codes of even dimension. *Designs, Codes and Cryptography*, 80(2):359–377, 2016.
- [26] David J. C. MacKay and Michael S. Postol. Weaknesses of margulis and ramanujan-margulis low-density parity-check codes. *Electr. Notes Theor. Comput. Sci.*, 74:97–104, 2002.
- [27] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 203–228. Springer, 2015.

- [28] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.
- [29] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. L.S.M. Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *IEEE International Symposium on Information Theory – ISIT’2013*, pages 2069–2073, Istanbul, Turkey, 2013. IEEE.
- [30] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986.
- [31] Alexander Nilsson, Thomas Johansson, and Paul Stankovski Wagner. Error amplification in code-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):238–258, November 2019.
- [32] Tom Richardson. Error floors of LDPC codes. In *Proc. of the 41th Annual Allerton Conf. on Communication, Control, and Computing*, 2003.
- [33] Jan Richter-Brockmann, Ming-Shing Chen, Santosh Ghosh, and Tim Güneysu. Racing BIKE: improved polynomial multiplication and inversion in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):557–588, 2022.
- [34] Jan Richter-Brockmann, Johannes Mono, and Tim Güneysu. Folding BIKE: Scalable hardware implementation for reconfigurable devices. Cryptology ePrint Archive, Report 2020/897, 2020. <https://eprint.iacr.org/2020/897>.
- [35] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018*, volume 10822 of *Lecture Notes in Computer Science*, pages 520–551. Springer, 2018.
- [36] Nicolas Sendrier. Decoding one out of many. In B.-Y. Yang, editor, *PQCrypto 2011*, volume 7071 of *LNCS*, pages 51–67. Springer, 2011.
- [37] Nicolas Sendrier. Secure sampling of constant-weight words – Application to BIKE. Cryptology ePrint Archive, Report 2021/1631, December 2021. <https://eprint.iacr.org/2021/1631>.
- [38] Nicolas Sendrier and Valentin Vasseur. Backflip: An improved QC-MDPC bit-flipping decoder. CBC 2019, 2019. <https://cbc2019.dii.univpm.it>.

- [39] Nicolas Sendrier and Valentin Vasseur. On the decoding failure rate of QC-MDPC bit-flipping decoders. In Jintai Ding and Rainer Steinwandt, editors, *PQCrypto 2019*, volume 11505 of *LNCS*, pages 404–416, Chongqing, China, May 2019. Springer.
- [40] Nicolas Sendrier and Valentin Vasseur. About low DFR for QC-MDPC decoding. In Jintai Ding and Jean-Pierre Tillich, editors, *PQCrypto 2020*, volume 12100 of *LNCS*, pages 20–34. Springer, 2020.
- [41] Nicolas Sendrier and Valentin Vasseur. On the existence of weak keys for QC-MDPC decoding. Cryptology ePrint Archive, Report 2020/1232, 2020.
- [42] Victor Shoup. Using hash functions as a hedge against chosen ciphertext attack. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 275–288. Springer, 2000.
- [43] Jean-Pierre Tillich. The decoding failure probability of MDPC codes. In *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*, pages 941–945, 2018.
- [44] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In Tsuyoshi Takagi, editor, *PQCrypto 2016*, volume 9606 of *LNCS*, pages 144–161. Springer, 2016.
- [45] Valentin Vasseur. *Post-quantum cryptography: a study of the decoding of QC-MDPC codes*. PhD thesis, Université de Paris, 2021.
- [46] Christof Zalka. Grover’s quantum searching algorithm is optimal. *Phys. Rev. A*, 60:2746–2751, October 1999.



# A Mathematical Background

## A.1 QC-MDPC Codes

**Definition 1** (Linear codes). A binary  $(n, k)$ -linear code  $\mathcal{C}$  of length  $n$  dimension  $k$  and co-dimension  $r = (n - k)$  is a  $k$ -dimensional vector subspace of  $\mathbb{F}_2^n$ .

**Definition 2** (Generator and Parity-Check Matrices). A matrix  $G \in \mathbb{F}_2^{k \times n}$  is called a generator matrix of a binary  $(n, k)$ -linear code  $\mathcal{C}$  if  $\mathcal{C} = \{mG \mid m \in \mathbb{F}_2^k\}$ . A matrix  $H \in \mathbb{F}_2^{(n-k) \times n}$  is called a parity-check matrix of  $\mathcal{C}$  if  $\mathcal{C} = \{c \in \mathbb{F}_2^n \mid cH^\top = 0\}$ .

**Definition 3** (Codeword and Syndrome). A codeword  $c \in \mathcal{C}$  of a vector  $m \in \mathbb{F}_2^{(n-k)}$  is  $c = mG$ . A syndrome  $s \in \mathbb{F}_2^r$  of a vector  $e \in \mathbb{F}_2^n$  is  $s = eH^\top$ .

### A.1.1 Circulant Matrices and Quasi-Cyclic Codes

A binary circulant matrix is a square matrix where each row is the rotation of one element to the right of the preceding row. It is completely defined by its first row. A block-circulant matrix is formed of circulant square blocks of identical size. The size of the circulant blocks is called the *order*. The *index* of a block-circulant matrix is the number of circulant blocks in a row.

**Definition 4** (Quasi-Cyclic Codes). A (binary) quasi-cyclic (QC) code of index  $n_0$  and order  $r$  is a linear code which admits as generator matrix a block-circulant matrix of order  $r$  and index  $n_0$ . A  $(n_0, k_0)$ -QC code is a quasi-cyclic code of index  $n_0$ , length  $n_0r$  and dimension  $k_0r$ .

### A.1.2 Circulant Matrices as a Polynomial Ring

There exists a natural ring isomorphism, denoted by  $\varphi$ , between the binary  $r \times r$  circulant matrices and the quotient polynomial ring  $\mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$ . The circulant matrix  $A$  whose first row is  $(a_0, \dots, a_{r-1})$  is mapped to the polynomial  $\varphi(A) = a_0 + a_1X + \dots + a_{r-1}X^{r-1}$ . This allows to view all matrix operations as polynomial operations. For every  $a = a_0 + a_1X + a_2X^2 + \dots + a_{r-1}X^{r-1}$  in  $\mathcal{R}$ , define  $a^\top = a_0 + a_{r-1}X + \dots + a_1X^{r-1}$ . This ensures  $\varphi(A^\top) = \varphi(A)^\top$ .

The mapping  $\varphi$  can be extended to any binary vector of  $\mathbb{F}_2^r$ . For all  $v = (v_0, v_1, \dots, v_{r-1})$ , set  $\varphi(v) = v_0 + v_1X + \dots + v_{r-1}X^{r-1}$ . It is easy to see that  $\varphi(vA) = \varphi(v)\varphi(A)$  and  $\varphi(vA^\top) = \varphi(v)\varphi(A)^\top$ .

**Factors of  $(X^r - 1)$ .** If  $r$  is even the scheme is subject to the squaring attack [25]. If  $r$  is divisible by  $2^\ell$ , the attack can be repeated  $\ell$  times and can reduce the security exponent. It is best to choose  $r$  odd, or even prime to thwart this attack. More generally, a good precaution is to choose  $r$  such that  $(X^r - 1) = (X - 1)f(X)$  where  $f(X)$  is an irreducible polynomial in  $\mathbb{F}_2[X]$  as this eliminates any potential structure in  $\mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$ . This happens when 2 is primitive modulo  $r$ .

**Invertible Elements.** An interesting side effect of choosing  $r$  such that 2 is primitive modulo  $r$  is that all elements of  $\mathcal{R}$  of odd weight are invertible in  $\mathcal{R}$ .

**Block-Circulant Matrices.** The block-circulant generator matrix of an  $(n_0, k_0)$ -QC code can be represented as a  $k_0 \times n_0$  matrix over  $\mathcal{R}$ . Each circulant block being represented by its image by  $\varphi$ . Similarly any parity check matrix can be viewed as an  $(n_0 - k_0) \times n_0$  matrix over  $\mathcal{R}$ . Respectively

$$G = \begin{pmatrix} g_{0,0} & \cdots & g_{0,n_0-1} \\ \vdots & & \vdots \\ g_{k_0-1,0} & \cdots & g_{k_0-1,n_0-1} \end{pmatrix}, H = \begin{pmatrix} h_{0,0} & \cdots & h_{0,n_0-1} \\ \vdots & & \vdots \\ h_{n_0-k_0-1,0} & \cdots & h_{n_0-k_0-1,n_0-1} \end{pmatrix}$$

with all  $g_{i,j}$  and  $h_{i,j}$  in  $\mathcal{R}$ . In all respects, a binary  $(n_0, k_0)$ -QC code can be viewed as an  $[n_0, k_0]$  code over the ring  $\mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$ . For instance the  $(2, 1)$  block-circulant matrix  $G = (G_0 \mid G_1)$  is represented by the  $1 \times 2$  matrix  $(g_0, g_1)$  over  $\mathcal{R}$  where  $g_0$  and  $g_1$  are the images of  $G_0$  and  $G_1$  by  $\varphi$  (*i.e.* the first row of  $G_0$  and  $G_1$ ).

### A.1.3 Definition of QC-MDPC Codes

A binary MDPC (Moderate-Density Parity-Check) code is a binary linear code which admits a somewhat sparse parity check matrix with a typical density of order  $O(\sqrt{n})$ . Such a matrix allows the use of iterative decoders similar to those used for LDPC (Low-Density Parity-Check) codes [18], widely deployed for error correction in telecommunication. QC-MDPC codes are formally defined as follows.

**Definition 5** (QC-MDPC code). *An  $(n_0, k_0, r, w)$ -QC-MDPC code is an  $(n_0, k_0)$  quasi-cyclic code of length  $n = n_0r$ , dimension  $k = k_0r$ , order  $r$  (and thus index  $n_0$ ) admitting a parity-check matrix with constant row weight  $w = O(\sqrt{n})$ .*

Gallager's bit flipping decoding [18] allows the efficient decoding of up to  $t = O(\sqrt{n})$  errors with high probability.

**Sparse Polynomials for BIKE.** The scheme makes use of  $(2, 1)$ -QC codes. Such codes are subspaces of  $\mathcal{R}^2$ . The private key  $(h_0, h_1) \in \mathcal{H}_w$  as in §1.2 defines the code

$$\mathcal{C} = \{(fh_1, fh_0) \mid f \in \mathcal{R}\} = \{(f_0, f_1) \in \mathcal{R}^2 \mid f_0h_0 + f_1h_1 = 0\}$$

with generator and parity check matrices (in  $\mathcal{R}^{1 \times 2}$ ) respectively

$$G = \begin{pmatrix} h_1 & h_0 \end{pmatrix} \text{ and } H = \begin{pmatrix} h_0^\top & h_1^\top \end{pmatrix}.$$

The corresponding binary matrices, as in Algorithm 1 for instance, are obtained by expanding the polynomials into circulant blocks.

## A.2 Decoding QC-MDPC Codes

### A.2.1 Decoding Algorithm for QC-MDPC Codes

The decoding of MDPC codes can be achieved, as for LDPC codes, with iterative decoders [18] and in particular with the (hard decision) bit flipping algorithm. Using floating point soft decision decoding would improve the decoding performance [4], but would also complexify the logic and the arithmetic, making the scheme less suitable for hardware and embedded device implementations, which is one of its interesting features [22]. Bit flipping decoding for QC-MDPC was suggested with the original design [29]. The decoding was later improved by a better threshold selection [9, 10], then by emulating soft decoding with the gray regions in the “One-Round” decoder of the round 1 BIKE proposal. Soft decision can also be emulated by giving a variable “time to live” to every flip in the decoding process [38, 40], this results in the Backflip decoder proposed for BIKE round 2. The Backflip decoder has a very low DFR but is not suitable for constant time implementation as shown in [14]. The latter work also shows that fine-tuned Black-Gray decoder is a better alternative (see §A.2.3) for a secure constant-time implementation.

### A.2.2 Decoding Algorithm for BIKE

The decoding algorithm (decoder) is a critical element of the decapsulation algorithm (Decaps) of BIKE. Its purpose is to find the unique solution of a decoding problem. During a key exchange session, the initiating party executes KeyGen and sends the public key to the responding party that is expected to send back some ciphertext. Subsequently, the initiating party executes Decaps, which, along with other steps specified in §2.2, invokes the decoding algorithm.

The decoder needs to be designed with the following targets: a) It has a sufficiently low DFR that satisfies the security requirements of the usage of the KEM; b) It runs a fixed number of steps; c) Its performance on the target platform is acceptable, and desirably high.

### A.2.3 Black-Gray Decoding

The authors of [14] discussed the importance of defining a decoder as an algorithm that runs a fixed number of steps (rather than a maximal number of steps). Such a definition also makes the algorithm implementable in constant-time, which is a required property from a cryptographic primitive. Of course, a real application needs to actually use a concrete constant-time implementation. In addition, [14] also identified the Black-Gray decoder as providing a favorable trade-off between: a) the number of steps; b) the estimated resulting DFR; c) the performance of a constant-time implementation. After a standard bit flipping iteration the Black-Gray decoder makes use, with several possible tuning, of two features introduced in the One-Round decoder of BIKE’s first round proposal: 1) a check of positions that were just flipped (black), then 2) a check of positions that were close but below the threshold (gray). The subsequent publication [16] by the same authors defined several variants of the Black-Gray decoder, and studied the resulting trade-offs. One variant is the Black-Gray-Flip (BGF) decoder that starts with one Black-Gray iteration and continues with several Bit-Flipping iterations. It was identified in [16] as the most efficient variant, at least for the studied platforms (see Algorithm 1 in [16]). BIKE uses the BGF decoder with tuned threshold functions that are based on fresh extensive simulations.

**Threshold Selection Rule**  $\text{threshold}(S, i)$ . The rule that is currently used derives from BIKE Round 1. In practice, for each security level it is given as an affine function of the syndrome weight. The numerical values are given in Section 2.3 for Level 1, 3, and 5. The coefficients of the current affine functions depend on the system parameters  $w$  and  $t$ , but not on  $r$  (see [45] for a comprehensive discussion). The current rule does not depend on the iteration number  $i$  either. Other strategies, depending on  $i$  and  $r$  are possible. Experiments indicate that those more elaborated strategies do not perform better (for the BGF decoder). Our simulations and estimated DFR claims are based on the rules given in the specification.

#### A.2.4 Estimating the DFR for High Block Size

**The Low Impact of Block Size on Computational Assumptions.** The block size  $r$  must be chosen large enough to allow efficient decoding. In practice one must choose  $r = \Omega(wt)$ . The higher  $r$  the lower the DFR. On the other hand, the best known attacks for codes of rate  $1/2$  as here, are of order  $2^{t(1+o(1))}$  or  $2^{w(1+o(1))}$ . This is corrected by a factor polynomial in  $r$  which is very small in practice. 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. This will simplify the extrapolation methodology described in the next paragraph.

**Estimating the DFR by Extrapolation.** Low DFR, *e.g.*,  $2^{-128}$ , as required for CCA security, cannot be directly estimated by simulation. Instead, simulations are combined with extrapolations, as described next. First, the DFR is measured for smaller block sizes  $r$ , for which simulations are meaningful (and assumed to provide a reliable estimation). Subsequently, one can define a curve based on the sample of  $r - DFR$  acquired values, and the curve is extrapolated to a larger block size for which the extrapolated DFR reaches the target. Known asymptotic models for simpler variants of bit flipping, as [43, 39], predict a concave shape for the curve in the relevant range of  $r$  values. Assuming a similar behavior, as described in [40], a linear extrapolation over two (acquired) points shoots to an overestimation of the required  $r$  (i.e., a conservative estimation). More extensive simulations can refine the DFR estimation and hence lead to smaller (more desirable) sufficient  $r$ . References [14] and [16] discuss simulation results with different extrapolations for several decoders, including the Black-Gray variants that are used for BIKE.

## B Known Attacks

### B.1 Hard Computational Problems

In the generic (*i.e.* non quasi-cyclic) case, the two following problems were proven NP-complete in [6].

**Problem 1** (Syndrome Decoding – SD).

Instance:  $H \in \mathbb{F}_2^{(n-k) \times n}$ ,  $s \in \mathbb{F}_2^{n-k}$ , an integer  $t > 0$ .

Property: *There exists*  $e \in \mathbb{F}_2^n$  such that  $|e| \leq t$  and  $eH^T = s$ .

**Problem 2** (Codeword Finding – CF).

Instance:  $H \in \mathbb{F}_2^{(n-k) \times n}$ , an integer  $t > 0$ .

Property: *There exists  $c \in \mathbb{F}_2^n$  such that  $|c| = t$  and  $cH^T = 0$ .*

In both problems the matrix  $H$  is the parity check matrix of a binary linear  $[n, k]$  code. Problem 1 corresponds to the decoding of an error of weight  $t$  and Problem 2 to the existence of a codeword of weight  $t$ . Both are also conjectured to be hard on average. This is argued in [1], together with results which indicate that the above problems remain hard even when the weight is very small, i.e.  $t = n^\varepsilon$ , for any  $\varepsilon > 0$ . Note that all known solvers for one of the two problems also solve the other and have a cost exponential in  $t$ .

### B.1.1 Hard Quasi-Cyclic Computational Problems

**Caveat.** In the sparse polynomial problems related to QC-MDPC codes, the parity of the weight matters. It doesn't make the problems easier or harder in practice but the weight parity of a sum, product, or inverse of elements of  $\mathcal{R}$  is determined by the weight parity of the operands. Those parities must be specified in problem statements and proofs, giving way to multiple versions. Stated versions are only those of interest for BIKE. The elements of  $\mathcal{R}$  of odd and even weight are respectively denoted  $\mathcal{R}_{\text{odd}}$  and  $\mathcal{R}_{\text{even}}$ . For any integer  $t$ , its parity is denoted  $p(t) \in \{\text{odd}, \text{even}\}$ .

**Problem 3** ((2, 1)-QC Syndrome Decoding – (2, 1)-QCSD).

Instance:  $(h, s) \in \mathcal{R}_{\text{odd}} \times \mathcal{R}_{p(t)}$ , an integer  $t > 0$ .

Property: *There exists  $(e_0, e_1) \in \mathcal{E}_t$  such that  $e_0 + e_1h = s$ .*

**Problem 4** ((2, 1)-QC Codeword Finding – (2, 1)-QCCF).

Instance:  $h \in \mathcal{R}_{\text{odd}}$ , an even integer  $w > 0$ , with  $w/2$  odd.

Property: *There exists  $(h_0, h_1) \in \mathcal{H}_w$  such that  $h_1 + h_0h = 0$ .*

The problems will be referred to respectively as QCSD $_{r,t}$  and QCCF $_{r,w}$ , indices being dropped unless an ambiguity is possible. A *witness* is an element, respectively of  $\mathcal{E}_t$  and  $\mathcal{H}_w$ , which verifies the property for some given input. The expression QCSD( $e, h, s$ ) is a boolean whose value is true if and only if  $e = (e_0, e_1)$  is a witness of QCSD for the input  $(h, s)$ , that is if  $e \in \mathcal{E}_t$  and  $e_0 + e_1h = s$ . For convenience, for any other input, including when the elements are out of range (*e.g.*  $e = \perp$ ), the value of QCSD( $e, h, s$ ) is false. Similarly, the expression QCCF( $h_0, h_1, h$ ) is true if and only if  $(h_0, h_1)$  is a witness of QCCF for the input  $h$ .

**Remark 4.** 1. *In the context of the general syndrome decoding problem, there is a search to decision reduction [2]. For the quasi-cyclic case, no such reduction is known.*

2. *Best known solvers for the quasi-cyclic problems above all derive from Information Set Decoding (ISD). Though these solvers are all designed for the search problems (i.e. find a witness for the instance), they do not perform essentially better for the decision problems (i.e. decide whether or not the property holds for the instance).*

**Key Security.** A key recovery adversary  $A$  against QCCF takes as argument  $h \in \mathcal{R}_{\text{odd}}$  and returns an element  $(h_0, h_1) \in \mathcal{H}_w \cup \{\perp\}$ . Its advantage is defined as

$$\text{Adv}_{\text{QCCF}}^{\text{OW}}(A) = \Pr \left[ \text{QCCF}(A(h_1 h_0^{-1}), h_1 h_0^{-1}) \mid (h_0, h_1) \stackrel{\$}{\leftarrow} \mathcal{H}_w \right].$$

A distinguisher  $D$  against QCCF takes as argument  $h \in \mathcal{R}_{\text{odd}}$  and returns true or false. Its advantage is defined as

$$\text{Adv}_{\text{QCCF}}^{\text{IND}}(D) = \left| \Pr \left[ D(h_1 h_0^{-1}) \mid (h_0, h_1) \stackrel{\$}{\leftarrow} \mathcal{H}_w \right] - \Pr \left[ D(h) \mid h \stackrel{\$}{\leftarrow} \mathcal{R}_{\text{odd}} \right] \right|.$$

**Message Security.** A (generic) decoder  $A$  against QCSD takes as argument  $(h, s) \in \mathcal{R}_{\text{odd}} \times \mathcal{R}_{\text{p}(t)}$  and returns  $e \in \mathcal{E}_t \cup \{\perp\}$ . Its advantage is defined as

$$\text{Adv}_{\text{QCSD}}^{\text{OW}}(A) = \Pr \left[ \text{QCSD}(A(h, e_0 + e_1 h), h, e_0 + e_1 h) \mid (h, (e_0, e_1)) \stackrel{\$}{\leftarrow} \mathcal{R}_{\text{odd}} \times \mathcal{E}_t \right].$$

Note that the requirement is that  $A(h, e_0 + e_1 h)$  returns a witness, not necessarily the error  $(e_0, e_1)$  used to build the instance. A distinguisher  $D$  against QCSD takes as argument  $(h, s) \in \mathcal{R}_{\text{odd}} \times \mathcal{R}_{\text{p}(t)}$  and returns true or false. Its advantage is defined as

$$\text{Adv}_{\text{QCSD}}^{\text{IND}}(D) = \left| \Pr \left[ D(h, e_0 + e_1 h) \mid (h, (e_0, e_1)) \stackrel{\$}{\leftarrow} \mathcal{R}_{\text{odd}} \times \mathcal{E}_t \right] - \Pr \left[ D(h, s) \mid (h, s) \stackrel{\$}{\leftarrow} \mathcal{R}_{\text{odd}} \times \mathcal{R}_{\text{p}(t)} \right] \right|.$$

Concretely, the hardness of Problems 3 and 4 is expressed by the fact that for all known adversaries of advantage  $\text{Adv}$ , for any of the above definitions, and running in time  $T$ , the quantity  $T/\text{Adv}$  grows exponentially with the instance size.

## B.2 Information Set Decoding

The best asymptotic variant of ISD is due to May and Ozerov [27], but it has a polynomial overhead which is difficult to estimate precisely. In practice, the BJMM

variant [5] is probably the best for relevant cryptographic parameters. The work factor for classical (*i.e.* non quantum) computing of any variant  $\mathcal{A}$  of ISD for decoding  $t$  errors (or finding a word of weight  $t$ ) in a binary code of length  $n$  and dimension  $k$  can be written

$$\text{WF}_{\mathcal{A}}(n, k, t) = 2^{ct(1+o(1))}$$

where  $c$  depends on the algorithm, on the code rate  $R = k/n$  and on the error rate  $t/N$ . It has been proven in [44] that, asymptotically, for sublinear weight  $t = o(n)$  (which is the case here as  $w \approx t \approx \sqrt{n}$ ),  $c = \log_2 \frac{1}{1-R}$  for all variants of ISD.

In practice, when  $t$  is small, using  $2^{ct}$  with  $c = \log_2 \frac{1}{1-R}$  gives a remarkably good estimate for the complexity. For instance, non asymptotic estimates derived from [21] give  $\text{WF}_{\text{BJMM}}(65542, 32771, 264) = 2^{263.3}$  “column operations” which is rather close to  $2^{264}$ . This closeness is expected asymptotically, but is circumstantial for fixed parameters. It only holds because various factors compensate, but it holds for most MDPC parameters of interest.

### B.2.1 Exploiting the Quasi-Cyclic Structure.

Both codeword finding and decoding are a bit easier (by a polynomial factor) when the target code is quasi-cyclic. If there is a word of weight  $w$  in a QC code then its  $r$  quasi-cyclic shifts are in the code. In practice, this gives a factor  $r$  speedup compared to a random code. Similarly, using Decoding One Out of Many (DOOM) [36] it is possible to produce  $r$  equivalent instances of the decoding problem. Solving those  $r$  instances together saves a factor  $\sqrt{r}$  in the workload. The system parameters will be chosen with the following guidelines:

- BIKE Message Security:  $\text{WF}(\text{QCSD}_{r,t}) \approx \frac{\text{WF}_{\text{ISD}}(2r, r, t)}{\sqrt{r}}$
- BIKE Key Security:  $\text{WF}(\text{QCCF}_{r,w}) \approx \frac{\text{WF}_{\text{ISD}}(2r, r, w)}{r}$

where  $\text{WF}(\text{QCSD}_{r,t})$  and  $\text{WF}(\text{QCCF}_{r,w})$  denote the average cost for finding a witness respectively to Problem 3 and Problem 4, and  $\text{WF}_{\text{ISD}}$  is the average cost of the best known ISD variant for the generic decoding of linear codes.

### B.2.2 Exploiting Quantum Computations.

Recall first that the NIST proposes to evaluate the quantum security as follows:



1. A quantum computer can only perform quantum computations of limited depth. They introduce a parameter, `MAXDEPTH`, which can range from  $2^{40}$  to  $2^{96}$ . This accounts for the practical difficulty of building a full quantum computer.
2. The amount (or bits) of security is not measured in terms of absolute time but in the time required to perform a specific task.

Regarding the second point, the NIST presents 6 security categories which correspond to performing a specific task. For example Task 1, related to Category 1, consists of finding the 128 bit key of a block cipher that uses AES-128. The security is then (informally) defined as follows:

**Definition 6.** *A cryptographic scheme is secure with respect to Category  $k$  iff. any attack on the scheme requires computational resources comparable to or greater than those needed to solve Task  $k$ .*

In what follows we will estimate that our scheme reaches a certain security level according to the NIST metric and show that the attack takes more quantum resources than a quantum attack on AES. We will use for this the following proposition.

**Proposition 1.** *Let  $f$  be a Boolean function which is equal to 1 on a fraction  $\alpha$  of inputs which can be implemented by a quantum circuit of depth  $D_f$  and whose gate complexity is  $C_f$ . Using Grover's algorithm for finding an input  $x$  of  $f$  for which  $f(x) = 1$  can not take less quantum resources than a Grover's attack on AES- $N$  as soon as*

$$\frac{D_f \cdot C_f}{\alpha} \geq 2^N D_{AES-N} \cdot C_{AES-N}$$

where  $D_{AES-N}$  and  $C_{AES-N}$  are respectively the depth and the complexity of the quantum circuit implementing AES- $N$ .

The point is that (essentially) the best quantum attack on our scheme consists in using Grover's search on the information sets computed in Prange's algorithm (this is Bernstein's algorithm [7]). Theoretically there is a slightly better algorithm consisting in quantizing more sophisticated ISD algorithms [24], however the improvement is tiny and the overhead in terms of circuit complexity make Grover's algorithm used on top of the Prange algorithm preferable in our case.

*Proof.* Following Zalka[46], the best way is to perform Grover's algorithm sequentially with the maximum allowed number of iterations in order not to go beyond `MAXDEPTH`. Grover's algorithm consists of iterations of the following procedure:

- Apply  $U : |0\rangle|0\rangle \rightarrow \sum_{x \in \{0,1\}^n} \frac{1}{2^{n/2}} |x\rangle |f(x)\rangle$ .
- Apply a phase flip on the second register to get  $\sum_{x \in \{0,1\}^n} \frac{1}{2^{n/2}} (-1)^{f(x)} |x\rangle |f(x)\rangle$ .
- Apply  $U^\dagger$ .

If we perform  $I$  iterations of the above for  $I \leq \frac{1}{\sqrt{\alpha}}$  then the winning probability is upper bounded by  $\alpha I^2$ . In our setting, we can perform  $I = \frac{\text{MAXDEPTH}}{D_f}$  sequentially before measuring, and each iteration costs time  $C_f$ . At each iteration, we succeed with probability  $\alpha I^2$  and we need to repeat this procedure  $\frac{1}{\alpha I^2}$  times to get a result with constant probability. From there, we conclude that the total complexity  $Q$  is:

$$Q = \frac{1}{\alpha I^2} \cdot I \cdot C_f = \frac{D_f \cdot C_f}{\alpha \text{MAXDEPTH}}. \quad (1)$$

A similar reasoning performed on using Grover's search on AES-N leads to a quantum complexity

$$Q_{AES-N} = \frac{2^N D_{AES-N} \cdot C_{AES-N}}{\text{MAXDEPTH}}. \quad (2)$$

The proposition follows by comparing (1) with (2).  $\square$

### B.3 Vulnerabilities Due to Decoding Failure

BIKE is currently designed to use ephemeral keys. However, if a keypair is reused, either inadvertently or by choice, the system is vulnerable to failure attacks.

#### B.3.1 The GJS Reaction Attack

The *reaction attack* [20] exploits correlation between the private key and the error patterns causing a failure. Collecting a few such error patterns allows an efficient key recovery attack. The amplification technique of [31] shows that, essentially, the discovery the first faulty error pattern dominates in the computational cost of the attack. And the average cost for discovering this first pattern is the inverse of the DFR. Hence, for instance, a DFR of  $2^{-128}$  is needed to ensure a 128 bits security level. Note that this is consistent with the IND-CCA security reduction (Theorem 3 of §C) and this also proves that the reduction is tight with respect to the DFR.

### B.3.2 Proving the DFR – Weak Keys and Error Floors

The decoding failure rate is defined on average over all private keys  $(h_0, h_1) \in \mathcal{H}_w$  and all errors  $(e_0, e_1) \in \mathcal{E}_t$ . This definition is relevant for the security reduction and thus for the IND-CCA security of BIKE.

Current methods to estimate the DFR are heuristic, see §A.2.4, and are based on simulations, extrapolations, and models for the decoder’s asymptotic behavior. But those models do not take into account all combinatorial properties of the codes. It is not possible to completely exclude the possibility that either particular codes (weak keys, as mentioned in [14]) or particular error patterns (near-codewords, leading to error floors, as mentioned in [40]) have a contribution to the average failure rate which is not captured by the extrapolation method.

**Weak Keys:** For any set of keys  $\mathcal{W} \subset \mathcal{H}_w$ , denote  $\text{DFR}(\mathcal{W})$  its relative DFR, taken on average over all errors and all keys in  $\mathcal{W}$ . If a set of keys is such that

$$\frac{\text{DFR}(\mathcal{W}) \cdot |\mathcal{W}|}{|\mathcal{H}_w|} > 2^{-\lambda},$$

then the (average) DFR would also be above the security requirement, even if the extrapolated failure rate was small enough. Such a set of weak keys was suggested in [14], with a relative DFR which was considerably higher than for a typical key. Later it was proven in [41] that, even when generalized, this family of weak keys had a negligible contribution to the average DFR.

**Error Floors:** Error floors happen in coding theory [26, 32] for some families of codes, including LDPC codes. They are caused by small weight words which also have syndromes of small weight.

## C A CCA Proof for BIKE

The BIKE protocol flows, as defined in §2, were proposed in [17]. Moreover, [17] showed that the flows conform to the HHK framework which consequently yielded the proof of IND-CCA security of BIKE (under the **Assumption 3.** on DFR). The proof in this appendix offers further details about the application of the HHK proofs to BIKE.

## C.1 An IND-CPA Proof for BIKE PKE

### C.1.1 From Computational Problems to OW-CPA

Table 10 gives a formal definition of the PKE of §1.2, denoted  $\text{PKE}_0$ . The OW-CPA

<b>KeyGen</b> <sub>0</sub>	Output: $(h_0, h_1) \in \mathcal{H}_w, h \in \mathcal{R}$ $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w ; h \leftarrow h_1 h_0^{-1}$
<b>Encrypt</b> <sub>0</sub>	Input: $h \in \mathcal{R}, (e_0, e_1) \in \mathcal{E}_t$ Output: $s \in \mathcal{R}$ $s \leftarrow e_0 + e_1 h$
<b>Decrypt</b> <sub>0</sub>	Input: $(h_0, h_1) \in \mathcal{H}_w, s \in \mathcal{R}$ Output: $e \in \mathcal{E}_t \cup \{\perp\}$ $e \leftarrow \text{decoder}(sh_0, h_0, h_1)$

Table 10:  $\text{PKE}_0$ : McEliece-like PKE from QC-MDPC codes

Game $G_3$ (OW-CPA)	Game $G_4$	$D(h)$ :
1: $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$	1: $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$	1:
2: $h \leftarrow h_1 h_0^{-1}$	2: $h \xleftarrow{\$} \mathcal{R}_{\text{odd}}$	2:
3: $(e_0^*, e_1^*) \xleftarrow{\$} \mathcal{E}_t$	3: $(e_0^*, e_1^*) \xleftarrow{\$} \mathcal{E}_t$	3: $(e_0^*, e_1^*) \xleftarrow{\$} \mathcal{E}_t$
4: $s^* \leftarrow e_0^* + e_1^* h$	4: $s^* \leftarrow e_0^* + e_1^* h$	4: $s^* \leftarrow e_0^* + e_1^* h$
5: $e \leftarrow A'(h, s^*)$	5: $e \leftarrow A'(h, s^*)$	5: $e \leftarrow A'(h, s^*)$
6: <b>return</b> $\text{QCSD}(e, h, s^*)$	6: <b>return</b> $\text{QCSD}(e, h, s^*)$	6: <b>return</b> $\text{QCSD}(e, h, s^*)$

Table 11: OW-CPA Security Games for  $\text{PKE}_0$

games for  $\text{PKE}_0$  are given in Table 11. The first game  $G_3$  is the standard OW-CPA game and the second game  $G_4$  is the generic decoding game. Recall that  $\text{QCSD}(e, h, s)$  is true if and only if  $e$  is a witness of  $\text{QCSD}$  (Problem 3) for the instance  $(h, s)$ . The advantage of those games for a given adversary  $A'$  is defined as the probability that the game outcome is true. Relevant computational problems are defined in §B.1.1.

**Theorem 1.** *For any OW-CPA adversary  $A'$  against  $\text{PKE}_0$  there exists a distinguisher  $D$  against  $\text{QCCF}$ , running in about the same time, such that*

$$\text{Adv}_{\text{PKE}_0}^{\text{OW-CPA}}(A') \leq \text{Adv}_{\text{QCCF}}^{\text{IND}}(D) + \text{Adv}_{\text{QCSD}}^{\text{OW}}(A').$$

*Proof.* 1. The difference between  $G_3$  and  $G_4$  lies solely on the way  $h$  is selected. The distinguisher  $D$  defined in Table 11 verifies

$$\begin{aligned}\text{Adv}^{G_3}(A') &= \Pr \left[ D(h_1 h_0^{-1}) \mid (h_0, h_1) \leftarrow^{\$} \mathcal{H}_w \right] \\ \text{Adv}^{G_4}(A') &= \Pr \left[ D(h) \mid h \leftarrow^{\$} \mathcal{R}_{\text{odd}} \right]\end{aligned}$$

and thus

$$|\text{Adv}^{G_3}(A') - \text{Adv}^{G_4}(A')| = \text{Adv}_{\text{QCCF}}^{\text{IND}}(D).$$

2. The adversary  $A'$  can be viewed as a decoder against QCSD. It verifies

$$\text{Adv}^{G_4}(A') = \text{Adv}_{\text{QCSD}}^{\text{OW}}(A').$$

Finally, since  $G_3$  is the OW-CPA game against  $\text{PKE}_0$

$$\text{Adv}^{G_3}(A') = \text{Adv}_{\text{PKE}_0}^{\text{OW-CPA}}(A') \leq \text{Adv}_{\text{QCCF}}^{\text{IND}}(D) + \text{Adv}_{\text{QCSD}}^{\text{OW}}(A')$$

□

### C.1.2 From OW-CPA to IND-CPA

Table 12 describes a new encryption scheme  $\text{PKE}$ , which is essentially a randomized version of  $\text{PKE}_0$ . It is constructed as a hybrid encryption scheme [42, 11], where the KEM part is derived from  $\text{PKE}_0$  as described in [12], and the DEM component is simply a one-time pad. Unlike  $\text{PKE}_0$ , the plaintext here is a bit-string  $m$ , and it is *not* embedded in the sparse vector  $(e_0, e_1)$ . It will be shown that  $\text{PKE}$  is IND-CPA secure, reducing tightly to the OW-CPA security of  $\text{PKE}_0$ . The hash function  $\mathbf{L}$  is modeled as random oracle for purpose of the proof, which is inspired again to [12].

The IND-CPA game  $G_0$  against  $\text{PKE}$  is given in Table 13. The advantage for an adversary  $A = (A_1, A_2)$  is defined as  $\text{Adv}_{\text{PKE}}^{\text{IND-CPA}}(A) = \text{Adv}^{G_0}(A) = |\Pr[G_0(A)] - 1/2|$ .

**Lemma 1.** *For any IND-CPA adversary  $A = (A_1, A_2)$  against  $\text{PKE}$ , there exists an OW-CPA adversary  $A'$  against  $\text{PKE}_0$ , running in about the same time, such that*

$$\text{Adv}_{\text{PKE}}^{\text{IND-CPA}}(A) \leq \frac{1}{2} \text{Adv}_{\text{PKE}_0}^{\text{OW-CPA}}(A')$$

<b>KeyGen<sub>0</sub></b>	Output: $(h_0, h_1) \in \mathcal{H}_w, h \in \mathcal{R}$ $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w ; h \leftarrow h_1 h_0^{-1}$
<b>Encrypt</b>	Input: $h \in \mathcal{R}, m \in \mathcal{M}$ Output: $c \in \mathcal{R} \times \mathcal{M}$ $(e_0, e_1) \xleftarrow{\$} \mathcal{E}_t ; c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$
<b>Decrypt</b>	Input: $(h_0, h_1) \in \mathcal{H}_w, (c_0, c_1)$ Output: $m \in \mathcal{M} \cup \{\perp\}$ $e \leftarrow \text{decoder}(sh_0, h_0, h_1)$ <b>if</b> $e = \perp$ <b>then</b> $m \leftarrow \perp$ <b>else</b> $m \leftarrow c_1 \oplus \mathbf{L}(e)$

Table 12: PKE: Randomization of PKE<sub>0</sub>

<b>Game <math>G_0</math> (IND-CPA)</b>	<b>Game <math>G_1^{(\dagger)}</math></b>	<b>Game <math>G_2^{(\dagger)}</math></b>
1: $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$	1: $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$	1: $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$
2: $h \leftarrow h_1 h_0^{-1}$	2: $h \leftarrow h_1 h_0^{-1}$	2: $h \leftarrow h_1 h_0^{-1}$
3: $e^* = (e_0^*, e_1^*) \xleftarrow{\$} \mathcal{E}_t$	3: $e^* = (e_0^*, e_1^*) \xleftarrow{\$} \mathcal{E}_t$	3: $e^* = (e_0^*, e_1^*) \xleftarrow{\$} \mathcal{E}_t$
4: $c_0^* \leftarrow e_0^* + e_1^* h$	4: $c_0^* \leftarrow e_0^* + e_1^* h$	4: $c_0^* \leftarrow e_0^* + e_1^* h$
5: $b \xleftarrow{\$} \{0, 1\}$	5: $b \xleftarrow{\$} \{0, 1\}$	5: $b \xleftarrow{\$} \{0, 1\}$
6: $(m_0^*, m_1^*, st) \leftarrow A_1(h)$	6: $(m_0^*, m_1^*, st) \leftarrow A_1(h)$	6: $(m_0^*, m_1^*, st) \leftarrow A_1(h)$
7: $c_1^* \leftarrow m_b^* \oplus \mathbf{L}(e^*)$	7: $c_1^* \leftarrow m_b^* \oplus \mathbf{L}(e^*)$	7: $c_1^* \xleftarrow{\$} \mathcal{M}$
8: $b' \leftarrow A_2(h, c_0^*, c_1^*, st)$	8: $b' \leftarrow A_2(h, c_0^*, c_1^*, st)$	8: $b' \leftarrow A_2(h, c_0^*, c_1^*, st)$
9: <b>return</b> $b = b'?$	9: <b>return</b> $b = b'?$	9: <b>return</b> $b = b'?$

<sup>(\dagger)</sup> game stops and returns true if  $A_1$  or  $A_2$  queries  $\mathbf{L}$  on a witness of QCS D for  $(h, c_0^*)$

Table 13: Games Sequence for the IND-CPA Security of PKE

*Proof.* 1. A first sequence of games for the proof is given in Table 13. The set of input queries to  $\mathbf{L}$  made by the adversary  $A$  is denoted  $\mathcal{L} \subset \mathcal{E}_t$ . The set

$$\mathcal{L}^* = \{e \in \mathcal{L} \mid \text{QCS D}(e, h, c_0^*)\}$$

denotes the set of witnesses of QCS D for the instance  $(h, c_0^*)$  queried by  $A$ . For convenience the event  $\mathcal{L}^* \neq \emptyset$  is denoted  $\mathcal{L}^*$ .

First remark that games  $G_0, G_1, G_2$  are identical when  $\mathcal{L}^* = \emptyset$ . It is clear for  $G_0$  and  $G_1$  because the stopping condition is never met. The distributions in  $G_2$  only differs for  $c_1^*$ :  $c_1^* \xleftarrow{\$} \mathcal{M}$  in  $G_2$  rather than  $c_1^* \leftarrow m_b^* \oplus \mathbf{L}(e^*)$  in  $G_0$  and  $G_1$ . If  $\mathcal{L}^* = \emptyset$  then, in particular,  $e^* \notin \mathcal{L}$  and  $\mathbf{L}(e^*)$  is never queried by the adversary. The value  $\mathbf{L}(e^*)$  is used once only, by the challenger, and is drawn uniformly at random in  $\mathcal{M}$  to emulate a random oracle for  $\mathbf{L}$ . Thus, if  $\mathcal{L}^* = \emptyset$

then  $c_1^* \leftarrow m_b^* \oplus \mathbf{L}(e^*)$  and  $c_1^* \xleftarrow{\$} \mathcal{M}$  yield distributions that are statistically indistinguishable. It follows that the probability of the event  $\mathcal{L}^*$  is identical in all games, the common value is denoted  $\Pr[\mathcal{L}^*]$ . Also

$$\Pr[G_0(A) \mid \neg\mathcal{L}^*] = \Pr[G_1(A) \mid \neg\mathcal{L}^*] = \Pr[G_2(A) \mid \neg\mathcal{L}^*] = \frac{1}{2}.$$

The last equality being true because  $b'$  is independent of  $b$  in  $G_2$ . Finally

$$\begin{aligned} \Pr[G_0(A)] &= \Pr[G_0(A) \wedge \neg\mathcal{L}^*] + \Pr[G_0(A) \wedge \mathcal{L}^*] \\ &\leq \Pr[G_0(A) \mid \neg\mathcal{L}^*] \cdot (1 - \Pr[\mathcal{L}^*]) + \Pr[\mathcal{L}^*] \\ &\leq \frac{1}{2} + \frac{1}{2} \Pr[\mathcal{L}^*]. \end{aligned}$$

Game $G'_3$	$\mathbf{L}'(e)$ :	Game $G_3$ (OW-CPA)
1: $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$	<b>if</b> QCSD( $e, h, c_0^*$ ) <b>then</b>	1: $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$
2: $h \leftarrow h_1 h_0^{-1}$	WT $\leftarrow e$	2: $h \leftarrow h_1 h_0^{-1}$
3: $e^* = (e_0^*, e_1^*) \xleftarrow{\$} \mathcal{E}_t$	<b>return</b> $\mathbf{L}(e)$	3: $(e_0^*, e_1^*) \xleftarrow{\$} \mathcal{E}_t$
4: $c_0^* \leftarrow e_0^* + e_1^* h$	$A'(h, c_0^*)$ :	4: $s^* \leftarrow e_0^* + e_1^* h$
5: $b \xleftarrow{\$} \{0, 1\}$ ; WT $\leftarrow \perp$	5: $b \xleftarrow{\$} \{0, 1\}$ ; WT $\leftarrow \perp$	5: $e \leftarrow A'(h, s^*)$
6: $(m_0^*, m_1^*, st) \leftarrow A_1^{\mathbf{L}'}(h)$	6: $(m_0^*, m_1^*, st) \leftarrow A_1^{\mathbf{L}'}(h)$	6: <b>return</b> QCSD( $e, h, s^*$ )
7: $c_1^* \xleftarrow{\$} \mathcal{M}$	7: $c_1^* \xleftarrow{\$} \mathcal{M}$	$A_1^{\mathbf{L}'}, A_2^{\mathbf{L}'}$ are as $A_1, A_2$ but call $\mathbf{L}'$ instead of $\mathbf{L}$
8: $b' \leftarrow A_2^{\mathbf{L}'}(h, c_0^*, c_1^*, st)$	8: $b' \leftarrow A_2^{\mathbf{L}'}(h, c_0^*, c_1^*, st)$	
9: <b>return</b> QCSD(WT, $h, c_0^*$ )	9: <b>return</b> WT	

Table 14: Adversary for PKE<sub>0</sub>

- The final step of the proof relates to OW-CPA games, see Table 14. The use of  $(A_1^{\mathbf{L}'}, A_2^{\mathbf{L}'})$  instead of  $(A_1, A_2)$  in  $G'_3$  does not change the distribution and only allows to maintain the variable WT. As argued earlier in the proof, the distributions in  $G'_3$  are identical to the distributions in  $G_2$  when  $\mathcal{L}^* = \emptyset$  and thus the probability of the event  $\mathcal{L}^* \neq \emptyset$  is the same here as in the earlier games, that is  $\Pr[\mathcal{L}^*]$ . The variable WT differs from  $\perp$  at the end of the game if and only if  $\mathcal{L}^* \neq \emptyset$ . If the variable WT differs from  $\perp$ , its value is a witness and the game succeeds, else it fails. Hence

$$\Pr[G'_3(A)] = \Pr[G'_3(A) \mid \mathcal{L}^*] \cdot \Pr[\mathcal{L}^*] = \Pr[\mathcal{L}^*]$$

It is readily observed that Game  $G'_3$  with the adversary  $A$  is identical to  $G_3$ , the OW-CPA game for  $\text{PKE}_0$ , with the adversary  $A'$ . Note that  $A'$  has access to  $(h, c_0^*)$ , thus it can use  $\mathbf{L}'$  to monitor the queries to  $\mathbf{L}$  and maintain the variable WT.

Putting everything together provides:

$$\text{Adv}_{\text{PKE}}^{\text{IND-CPA}}(A) = \left| \Pr[G_0(A)] - \frac{1}{2} \right| \leq \frac{1}{2} \Pr[\mathcal{L}^*] = \frac{1}{2} \Pr[G_3(A')] = \frac{1}{2} \text{Adv}_{\text{PKE}_0}^{\text{OW-CPA}}(A').$$

□

**Theorem 2.** *For any IND-CPA adversary  $A = (A_1, A_2)$  against PKE there exists a distinguisher  $D$  against QCCF and a decoder  $A'$  against QCSD, both running in about the same time as  $A$ , such that*

$$\text{Adv}_{\text{PKE}}^{\text{IND-CPA}}(A) \leq \frac{1}{2} \text{Adv}_{\text{QCCF}}^{\text{IND}}(D) + \frac{1}{2} \text{Adv}_{\text{QCSD}}^{\text{OW}}(A').$$

*Proof.* The proof simply combines Theorem 1 and Lemma 1. □

## C.2 From IND-CPA to IND-CCA

### C.2.1 PKE Correction and DFR

In [23], PKE is defined to be  $\delta$ -correct if

$$\mathbb{E}[\max_{m \in \mathcal{M}} \Pr[\text{Decrypt}(sk, c) \neq m \mid c \leftarrow \text{Encrypt}(pk, m)]] \leq \delta \quad (3)$$

where the expectation is taken over  $(pk, sk) \leftarrow \text{KeyGen}^{\text{PKE}}$ . Failure to decrypt a ciphertext  $(c_0 = e_0 + e_1 h, c_1)$  happens if and only if  $(e_0, e_1) \neq \text{decoder}(c_0 h_0, h_0, h_1)$  and  $c_0$  only depends on the internal randomness, not on the message  $m$ . This property is referred to as *message-agnostic* in [17]. The max vanishes in (3). It follows that PKE is  $\delta$ -correct for any  $\delta$  such that

$$\Pr[(e_0, e_1) \neq \text{decoder}(e_0 h_0 + e_1 h_1, h_0, h_1) \mid (h_0, h_1) \xleftarrow{\$} \mathcal{H}_w, (e_0, e_1) \xleftarrow{\$} \mathcal{E}_t] \leq \delta$$

The left-hand side above is precisely the DFR as defined in the setup.

### C.2.2 HHK Proof

The proof framework of [23] transforms a probabilistic public-key encryption scheme, here PKE (Table 12), first into a derandomized variant  $\text{PKE}_1$  (Table 15) then into a key encapsulation mechanism with implicit rejection  $\text{KEM}^\neq$  (Table 16). A hash function  $\mathbf{H}$  is required for  $\text{PKE}_1$  and another one  $\mathbf{K}$  for  $\text{KEM}^\neq$ .



<b>KeyGen<sub>0</sub></b>	Output: $(h_0, h_1) \in \mathcal{H}_w, h \in \mathcal{R}$ $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w; h \leftarrow h_1 h_0^{-1}$
<b>Encrypt<sub>1</sub></b>	Input: $h \in \mathcal{R}, m \in \mathcal{M}$ Output: $c \in \mathcal{R} \times \mathcal{M}$ $(e_0, e_1) \leftarrow \mathbf{H}(m); c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$
<b>Decrypt<sub>1</sub></b>	Input: $(h_0, h_1) \in \mathcal{H}_w, c \in \mathcal{R} \times \mathcal{M}, h \in \mathcal{R}$ Output: $m \in \mathcal{M} \cup \{\perp\}$ $m \leftarrow \mathbf{Decrypt}((h_0, h_1), c)$ <b>if</b> $m \neq \perp$ <b>and</b> $c \neq \mathbf{Encrypt}_1(h, m)$ <b>then</b> $m \leftarrow \perp$

Table 15: PKE<sub>1</sub>: Derandomizing PKE

<b>KeyGen</b>	Output: $(h_0, h_1, \sigma) \in \mathcal{H}_w \times \mathcal{M}, h \in \mathcal{R}$ $(h_0, h_1, \sigma) \xleftarrow{\$} \mathcal{H}_w \times \mathcal{M}; h \leftarrow h_1 h_0^{-1}$
<b>Encaps</b>	Input: $h \in \mathcal{R}$ Output: $K \in \mathcal{K}, c \in \mathcal{R} \times \mathcal{M}$ $m \xleftarrow{\$} \mathcal{M}; c \leftarrow \mathbf{Encrypt}_1(h, m); K \leftarrow \mathbf{K}(m, c)$
<b>Decaps</b>	Input: $(h_0, h_1, \sigma) \in \mathcal{H}_w \times \mathcal{M}, c \in \mathcal{R} \times \mathcal{M}, h \in \mathcal{R}$ Output: $K \in \mathcal{K}$ $m \leftarrow \mathbf{Decrypt}_1((h_0, h_1), c, h)$ <b>if</b> $m \neq \perp$ <b>then</b> $K \leftarrow \mathbf{K}(m, c)$ <b>else</b> $K \leftarrow \mathbf{K}(\sigma, c)$

Table 16: KEM<sup>≠</sup>: KEM with Implicit Rejection From PKE<sub>1</sub>

**Lemma 2.** [23, §3.3] *If PKE is  $\delta$ -correct, for all IND-CCA adversary  $B$  against KEM<sup>≠</sup> issuing at most  $q$  queries to  $\mathbf{K}$  or  $\mathbf{H}$ , there exists an IND-CPA adversary  $A$  against PKE, running in about the same time, such that*

$$\text{Adv}_{\text{KEM}^\neq}^{\text{IND-CCA}}(B) \leq q \cdot \delta + \frac{3 \cdot q}{|\mathcal{M}|} + 3 \cdot \text{Adv}_{\text{PKE}}^{\text{IND-CPA}}(A).$$

*Proof (sketch).* There are two key theorems in [23] to prove that KEM<sup>≠</sup> is IND-CCA secure. One relates the OW-PCVA security of PKE<sub>1</sub> to the IND-CPA security of PKE. The other relates the OW-PCA security of PKE<sub>1</sub> with the IND-CCA security of KEM<sup>≠</sup>. [23, Theorem 3.4] states that for all IND-CCA adversaries  $B$  against KEM<sup>≠</sup>, issuing at most  $q_K$  queries to  $\mathbf{K}$ , there exists an OW-PCA adversary  $B'$  against PKE<sub>1</sub>, running in about the same time, issuing at most  $q_K$  queries to PCO such that

$$\text{Adv}_{\text{KEM}^\neq}^{\text{IND-CCA}}(B) \leq \frac{q_K}{|\mathcal{M}|} + \text{Adv}_{\text{PKE}_1}^{\text{OW-PCA}}(B')$$

PCO is a plaintext checking oracle which returns true on input  $(m, c)$  if and only if  $m = \mathbf{Decrypt}_1((h_0, h_1), c)$ . [23, Theorem 3.2] is stated for an OW-PCVA adversary, which is a stronger concept than the OW-PCA. A simpler version, downgraded to OW-PCA is stated here. If PKE is  $\delta$ -correct, [23, Theorem 3.2] states that for all OW-PCA adversary  $B'$  against  $\text{PKE}_1$  issuing at most  $q_H$  queries to  $\mathbf{H}$  and  $q_K$  queries to PCO, there exists an IND-CPA adversary  $A$  against PKE, running in about the same time, such that

$$\text{Adv}_{\text{PKE}_1}^{\text{OW-PCA}}(B') \leq q_H \cdot \delta + \frac{2 \cdot q_H + 1}{|\mathcal{M}|} + 3 \cdot \text{Adv}_{\text{PKE}}^{\text{IND-CPA}}(A)$$

Lemma 2 is proved by combining the two results. The total number of queries to random oracles  $\mathbf{H}$  and  $\mathbf{K}$  is aggregated to  $q$ .  $\square$

### C.3 The BIKE Key Encapsulation Mechanism

The construction proposed in [23] is convenient because it provides easy tools for the proof. To obtain a self-contained description of  $\text{KEM}^\perp$ , the nested calls of Table 16 must be inlined. The result is the KEM described in Table 17, which is precisely the one given in this specification in §2.2.

<p><b>KeyGen</b> : <math>() \mapsto (h_0, h_1, \sigma), h</math>  Output: <math>(h_0, h_1, \sigma) \in \mathcal{H}_w \times \mathcal{M}, h \in \mathcal{R}</math>  1: <math>(h_0, h_1) \xleftarrow{\\$} \mathcal{H}_w</math>  2: <math>h \leftarrow h_1 h_0^{-1}</math>  3: <math>\sigma \xleftarrow{\\$} \mathcal{M}</math></p>	<p><b>Encaps</b> : <math>h \mapsto K, c</math>  Input: <math>h \in \mathcal{R}</math>  Output: <math>K \in \mathcal{K}, c \in \mathcal{R} \times \mathcal{M}</math>  1: <math>m \xleftarrow{\\$} \mathcal{M}</math>  2: <math>(e_0, e_1) \leftarrow \mathbf{H}(m)</math>  3: <math>c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))</math>  4: <math>K \leftarrow \mathbf{K}(m, c)</math></p>
<p><b>Decaps</b> : <math>(h_0, h_1, \sigma), c \mapsto K</math>  Input: <math>((h_0, h_1), \sigma) \in \mathcal{H}_w \times \mathcal{M}, c = (c_0, c_1) \in \mathcal{R} \times \mathcal{M}</math>  Output: <math>K \in \mathcal{K}</math>  1: <math>e' \leftarrow \text{decoder}(c_0 h_0, h_0, h_1) \quad \triangleright e' \in \mathcal{R}^2 \cup \{\perp\}</math>  2: <math>m' \leftarrow c_1 \oplus \mathbf{L}(e') \quad \triangleright</math> with the convention <math>\perp = (0, 0)</math>  3: <b>if</b> <math>e' = \mathbf{H}(m')</math> <b>then</b> <math>K \leftarrow \mathbf{K}(m', c)</math> <b>else</b> <math>K \leftarrow \mathbf{K}(\sigma, c)</math></p>	

Table 17:  $\text{KEM}^\perp$  Inlined from Tables 12, 15, and 16

The transformation is straightforward and the decapsulation can be simplified by adding the convention  $\perp = (0, 0) \in \mathcal{R}^2$ , so that  $\mathbf{L}(\perp)$  is meaningful, and by

remarking that since the range of  $\mathbf{H}$  is  $\mathcal{E}_t$ , checking  $e' = \mathbf{H}(m')$  also checks  $|e'| = t$  and  $e' \neq \perp = (0, 0)$ .

**Theorem 3.** *If PKE is  $\delta$ -correct, for all IND-CCA adversary  $B$  against  $\text{KEM}^\perp$  issuing at most  $q$  queries to  $\mathbf{K}$  or  $\mathbf{H}$ , there exists a distinguisher  $D$  against QCCF and a decoder  $A'$  against QCSD, both running in about the same time as  $B$ , such that*

$$\text{Adv}_{\text{KEM}^\perp}^{\text{IND-CCA}}(B) \leq q \cdot \delta + \frac{3 \cdot q}{|\mathcal{M}|} + \frac{3}{2} \text{Adv}_{\text{QCCF}}^{\text{IND}}(D) + \frac{3}{2} \text{Adv}_{\text{QCSD}}^{\text{OW}}(A'). \quad (4)$$

*Proof.* The proof combines Theorem 2 and Lemma 2. □

### C.3.1 Concrete Security and Parameters Selection

To offer  $\lambda$  bits of security it is typically required that  $|A|/\text{Adv}(A) \geq 2^\lambda$  for all adversaries  $A$  running in time  $|A|$ . Observing that the running time must exceed the number of oracle queries, it follows from Theorem 3 that  $\text{KEM}^\perp$ , the BIKE key encapsulation mechanism, offers  $\lambda$  bits of (classical) security in the IND-CCA game if the system parameters  $r, w, t, \ell$ , and decoder are selected at setup such that

1.  $\text{QCCF}_{r,w}$  offers  $\lambda$  bits of security
2.  $\text{QCSD}_{r,t}$  offers  $\lambda$  bits of security
3.  $|\mathcal{M}| = 2^\ell \geq 2^\lambda$
4.  $\text{DFR}(\text{decoder}) \leq 2^{-\lambda}$ .

Note that if all conditions are met except condition 4 on the DFR, the scheme is still IND-CPA secure.

The computational problems guide the selection of  $w$  and  $t$  (and not  $r$ ) based on the best known solvers, as discussed in §B.2, and on the fact that the block size  $r$  has a very limited influence on those solvers' complexity (see §A.2.4). Choosing  $\ell$  large enough is straightforward. Last, with  $w$  and  $t$  fixed, the block length  $r$  is selected so that the DFR estimate is low enough, as discussed in §A.2.4.

There are additional requirements for the parameters selection (see §A.1.2): 1) the block size is chosen such that 2 is primitive mod  $r$  to avoid any undesirable structure in the polynomial ring  $\mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$ , and 2) the row weight  $w$  is chosen even and such that  $|h_0| = |h_1| = w/2$  is odd to ensure that  $h_0$  is always invertible in  $\mathcal{R}$ .

## C.4 Security with Non Uniform Key Generation

Two games related to BIKE are considered in Table 18 which differ only by the way the secret key is sampled, either uniformly or using some distribution  $\mathcal{D}$  over  $\mathcal{H}_w$ .

Game $G$ (biased key)	Game $G'$ (uniform key)
1: $(h_0, h_1) \stackrel{\mathcal{D}}{\leftarrow} \mathcal{H}_w$	1: $(h_0, h_1) \stackrel{\$}{\leftarrow} \mathcal{H}_w$
$\vdots$ same game	$\vdots$ same game

Table 18: Biased *vs* Uniform Key Sampling

**Claim:** if an adversary  $A$  is able to win the (biased) game  $G$  with a success probability  $\varepsilon$ , then the same adversary will win the game  $G'$  with probability  $\leq \tau\varepsilon$  where

$$\tau = \max_{h_0, h_1} \frac{\Pr \left[ (h_0, h_1) \mid (h_0, h_1) \stackrel{\mathcal{D}}{\leftarrow} \mathcal{H}_w \right]}{\Pr \left[ (h_0, h_1) \mid (h_0, h_1) \stackrel{\$}{\leftarrow} \mathcal{H}_w \right]}.$$

Proving this claim is straightforward (see [37, §4.2, equation (5)] for instance). Now, in all the security games related to public key encryption and key encapsulation mechanism which are mentioned in this section a *single* key pair is drawn at the beginning of the game, thus a single usage of distribution  $\mathcal{D}$  is made. No adversary can gain more than a factor  $\tau$  for its probability of success when a biased key is used rather than a uniform one. Finally, if the distribution  $\mathcal{D}$  consists in drawing  $h_0$  and  $h_1$  with two calls to Algorithm 3, both with  $(\text{len}, \text{wt}) = (r, w/2)$ , the value of  $\tau$ , which derives from [37, Proposition 3], is

$$\tau \leq 2 \prod_{i=0}^{w/2-1} \left( 1 + \frac{(r-i) - r_i}{2^{32}} \right), r_i = 2^{32} \pmod{(r-i)}, 0 \leq i < w/2.$$

Security	$r$	$w$	$t$	$\tau$
Level 1	12,323	142	134	$\tau \leq 1.00023$
Level 3	24,659	206	199	$\tau \leq 1.00059$
Level 5	40,973	274	264	$\tau \leq 1.00130$