BIKE:
Bit Flipping Key Encapsulation

Round 3 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

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: Submitters, Amazon Web Services, Intel Corporation, Worldline.

Email Address (preferred): rafaelmisoczki@google.com

Postal Address (if absolutely necessary):
Rafael Misoczki, Google, 803 11th Avenue, Sunnyvale, CA 94089.

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

Version: 4.1

Release Date: 10/22/2020

Tweaks in BIKE between Rounds 2 and 3

The main tweaks incorporated to the BIKE proposal between Rounds 2 and 3 are:

- **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 ParallelHash procedure by normal hashing (SHA384), leading to superior performance for our reference implementation. See Section 2.5.

Contents

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 $\{(h_0, h_1) \in \mathcal{R}^2 \mid \lvert h_0\rvert = \lvert h_1\rvert = w/2\}$
$\mathcal{E}_t$:	Error space $\{(e_0, e_1) \in \mathcal{R}^2 \mid \lvert e_0\rvert + \lvert e_1\rvert = t\}$
$\lvert g\rvert$:	Hamming weight of a binary polynomial $g \in \mathcal{R}$.
$u \xleftarrow{\$} U$:	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

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

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 [5, 1], taking the form of distinguishing problems as given in Table 2.

Private Key: $(h_0, h_1) \in \mathcal{H}_w$	**Encryption:** $(e_0, e_1) \in \mathcal{E}_t \mapsto s = e_0 + e_1 h \in \mathcal{R}$
Public Key: $h = h_1 h_0^{-1} \in \mathcal{R}$	**Decryption:** $s \in \mathcal{R} \mapsto \text{decoder}(sh_0, h_0, h_1) \in \mathcal{E}_t$

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

Key:	distinguish $h_1 h_0^{-1}$ from random, $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$
Message:	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 λ.
Output: Parameters $\{r, w, t, \ell\}$, hash functions $\{\mathbf{H}, \mathbf{K}, \mathbf{L}\}$, and `decoder`.

1. **System Parameters.** Select r, w, t, ℓ 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.
   - ℓ (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.

   - $\mathbf{H} : \mathcal{M} \to \mathcal{E}_t$.
   - $\mathbf{K} : \mathcal{M} \times \mathcal{R} \times \mathcal{M} \to \mathcal{K}$.
   - $\mathbf{L} : \mathcal{R}^2 \to \mathcal{M}$

   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 $\texttt{decoder}(s, h_0, h_1)$ 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

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

   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

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

Table 3: The BIKE Key Encapsulation Mechanism

2.3 Decoder

The selected decoder is the Black-Gray-Flip (BGF) defined in [14]. 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 τ, 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, τ, and of the threshold function must be specified for every parameter set. The algorithm invokes two functions specified as follows:

- $\texttt{ctr}(H, s, j)$. 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.

- $\texttt{threshold}(S, i)$. This function is the threshold selection rule. It depends, in general, on the syndrome weight S, the iteration number i, and on the system

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

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

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

$e \leftarrow 0^n$
for $i = 1, \ldots,$ NbIter **do**
    $T \leftarrow \texttt{threshold}(|s + eH^\intercal|, i)$
    $e, \text{black}, \text{gray} \leftarrow \texttt{BFIter}(s + eH^\intercal, e, T, H)$
    if $i = 1$ **then**
        $e \leftarrow \texttt{BFMaskedIter}(s + eH^\intercal, e, \text{black}, (d+1)/2 + 1, H)$
        $e \leftarrow \texttt{BFMaskedIter}(s + eH^\intercal, e, \text{gray}, (d+1)/2 + 1, H)$
if $s = eH^\intercal$ **then**
    return e
else
    return $\perp$

`procedure BFIter`(s, e, T, H)
for $j = 0, \ldots, n-1$ **do**
    if $\texttt{ctr}(H, s, j) \geq T$ **then**
        $e_j \leftarrow e_j \oplus 1$
        $\text{black}_j \leftarrow 1$
    else if $\texttt{ctr}(H, s, j) \geq T - \tau$ **then**
        $\text{gray}_j \leftarrow 1$
return $e, \text{black}, \text{gray}$

`procedure BFMaskedIter`$(s, e, \text{mask}, T, H)$
for $j = 0, \ldots, n-1$ **do**
    if $\texttt{ctr}(H, s, j) \geq T$ **then**
        $e_j \leftarrow e_j \oplus \text{mask}_j$
return e

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 [14]. 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 odd weight (Algorithm 3).
- With a specific weight w (Algorithm 4).

Remark 1. *Algorithm 4 is a "Rejection Sampling" method. It 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 $|U| = w$).*

AES-CTR based pseudorandom bits generation. The building block for these algorithms is AES-256 (256 bits key) in CTR mode using a 96-bit zero IV ($\text{IV} = 0^{96}$) and a 32-bit counter starting from 0^{32}. Suppose that seed is a 256-bit key and μ is a positive integer. Denote the μ blocks (of 128 bits each) output of AES-CTR with that key by AES-CTR (seed, μ). Let ν be a positive integer. Then, the least significant ν bits of AES-CTR (seed, $1 + \mathsf{floor}\,((\nu/128))$) are denoted by AES-CTR-Stream (seed, ν).

Algorithm 2 GenPseudoRand(seed, len)

Require: seed (32 bytes)

1: **return** AES-CTR-Stream (seed, len)

Algorithm 3 GenPseudoRandOddWeight(seed, len)

Require: seed (32 bytes), len

1: z = GenPseudoRand(seed, len)

2: **if** $|z|$ is even **then** $z[0] = z[0] \oplus 1$ ▷ $z[0]$ is the least significant bit of z

3: **return** z

Algorithm 4 WAES-CTR-PRF(s, wt, len)

Require: wt (32 bits), len

Ensure: A list (wlist) of wt bit-positions in $[0, \ldots, \text{len} - 1]$.

1: wlist$= \phi$; $ctr = 0$; $i = 0$

2: s = AES-CTR-Stream(seed, ∞) ▷ ∞ denotes "sufficiently large"

3: $mask = (2^{\text{ceil}(\log_2 r)} - 1)$

4: **while** $ctr <$ wt **do**

5: pos $= s[32(i+1) - 1 : 32i]$ & $mask$ ▷ & denotes bitwise AND

6: **if** ((pos $<$ len) AND (pos $\notin$ wlist)) **then**

7: wlist $=$ wlist $\cup \{\text{pos}\}$; $ctr = ctr + 1$; $i = i + 1$

8: **return** wlist, s

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 ℓ bits that is input to the function. It is generated by invoking Algorithm 4 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†
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.

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

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 $\mathrm{QCSD}_{r,t}$

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

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 [20] where a KEM is δ-correct if the decapsulation fails (*i.e.* disagrees with encapsulation) with probability at most δ on average over all keys and messages. Similarly, a decoder will be δ-correct if its failure rate is at most δ 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 [20] is also valid in the QROM. Tightness was later improved in [33]. 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 [17] 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 λ bits of (classical) IND-CCA security, the parameters are r, w, t, ℓ, and `decoder` chosen in the setup such that:

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

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 [17] 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 [40] and Markovian model of bit flipping [36]. 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 [12], they have a low density, but have a strong impact on decoding. Work is in progress on this matter, and preliminary results [38] give an indication that, even generalized, those weak keys are too few and have a negligible impact on the average DFR.

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 [37] 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 $(h_0^{\mathsf{T}}, 0)$ is h_0^2, and both the word and syndrome have weight $w/2$. Work is in progress on this matter, and so far there is no indication that the contribution of error patterns close to near-codewords is dominant in the average DFR.

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 guarantied 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 [15]. 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 [40, 36].

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 [17] 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 ($\text{FO}^{\not\perp}$, as described in [20]) for converting a δ-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[1].

[1] `https://bikesuite.org`

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 $\text{FO}^{\not\perp}$ construction of [20]. Further modifications, the shared secret derivation and the domain separation of the hash functions $\mathbf{H}, \mathbf{K}, \mathbf{L}$, were made in [15] to match precisely the $\text{FO}^{\not\perp}$ construction of [20] 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 $\text{FO}^{\not\perp}$ transformation and targets only IND-CPA security. However, the difference in the performance is negligible (see [12]) 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.

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 [13] 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 [13]. 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, [13] 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 was the pseudorandom generation determined?

The pseudorandom generation uses the standard AES-CTR with a 256-bit key. It is very efficient on modern processors that have dedicated AES instructions (e.g., AES-NI). In all cases the generated pseudorandom stream is short enough to ignore the incremental distinguishing advantage in the security analysis of the scheme.

4.3.9 How were the functions $\mathbf{H}, \mathbf{K}, \mathbf{L}$ designed?

BIKE specification models $\mathbf{H}, \mathbf{K}, \mathbf{L}$ as random oracles. The concrete realization of $\mathbf{K}$ and $\mathbf{L}$ relies on the standard SHA384 hash function that has sufficient capacity in its compression function, and is accepted by NIST for this purpose. The function $\mathbf{H}$ uses 256 bits as a key, and AES-CTR based pseudorandom expansion.

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 (σ). 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, (σ), requires ℓ 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[2]. 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[3]. The package includes the following implementations:

1. PORTABLE: a C (C99) portable code implementation.
2. AVX2: implementation that leverages the AVX2 architecture features. It is written in C (with C intrinsics for AVX2 functions).

[2] `https://bikesuite.org/reference.html`
[3] `https://github.com/awslabs/bike-kem`

3. AVX512: implementation that leverages the AVX512 architecture features. It is written in C (with C intrinsics for AVX512 functions). This implementation can also be compiled to use the latest `vector-PCLMULQDQ` instruction that is available on the Intel IceLake processors.

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.

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 [11].

The benchmarking platform. The platform used in the experiments was equipped with 10^{th} generation Intel®Core^{TM} processor (microarchitecture codename "Ice Lake"[ICL]). The machine is Dell XPS 13 7390 2in1 laptop with Intel®Core^{TM} i7-1065G7 CPU working at 1.30GHz, with 16 GB RAM, 48K L1d cache, 32K L1i cache, 512K L2 cache, and 8MiB L3 cache. The CPU supports AVX512 instruction set and `vector-PCLMULQDQ` instruction. 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 gcc (version 9.2.1) and ran on a Linux OS (Ubuntu 19.04).

Performance numbers

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

	AVX2	AVX512	VPCLMUL
KeyGen	600	585	470
Encaps	220	205	195
Decaps	2220	1356	1280

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

	AVX2	AVX512	VPCLMUL
KeyGen	1780	1760	1280
Encaps	465	435	410
Decaps	6610	3825	3500

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 AVX2 and AVX512 support. This property is due to the* `PCLMULQDQ` *instruction, and even more so to the newer* `vector-PCLMULQDQ` *instruction.*

5.4 Hardware Implementation

The Hardware Implementation Code for BIKE was developed by Jan Richter-Brockmann and Tim Güneysu. The hardware implementation includes

1. Reference implementation for Key Generation Level 1
2. Reference implementation for Encapsulation Level 1
3. Reference implementation for Decapsulation Level 1

All the hardware files are published on the BIKE website[4].

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 classic ITA [23] and a slightly adapted version of Algorithm 1 defined in [21]. The underlying squaring operations are optimal allocated to a unit which either performs arbitrary exponentiations or simple squarings. For the encapsulation we utilize a slightly improved version of the multiplier proposed in [22]. Eventually, the decapsulation is accomplished by a hardware implementation of the BGF decoder. By instantiating multiple UPC

[4] `https://bikesuite.org/`

Table 9: Implementation results of the decapsulation module for Level 1 ($r = 12\,323$).

	Resources					Performance		
	Logic		Memory		Area	Cycles	Frequency	Latency
	LUT	DSP	FF	BRAM	Slices	Cycles (average)	MHz	ms (average)
KeyGen	12 654	0	1 044	10	3 554	258 750	96.15	2.69
Encaps	14 894	0	3 477	10	4 313	12 240	121.95	0.10
Decaps	29 908	13	5 075	29	8 610	189 615	100	1.90

equation counter in parallel, the decoding can be highly accelerated. Note that the hardware utilization of the encapsulation and decapsulation module include the instantiations of the random oracles, i.e., an AES-256 and a SHA384.

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-Level1_3114.req`
- rsp file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level1_3114.rsp`
- req file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level3_6198.req`
- rsp file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level3_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 [12] (accepted to CBCrypto 2020), [14] (accepted to PQCrypto 2020), [13] (accepted to CSCML 2020), [15]. They are also co-developers of the Additional Software Implementation Code Package of BIKE (residing in the git repository `github.com/awslabs/bike-kem`).

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] 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.

[4] 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.

[5] 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.

[6] Daniel J Bernstein. Grover vs. McEliece. In *International Workshop on Post-Quantum Cryptography*, pages 73–80. Springer, 2010.

[7] 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.

[8] Tung Chou. Qcbits: Constant-time small-key code-based cryptography. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 280–300. Springer, 2016.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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`.

[14] 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.

[15] 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.

[16] R. G. Gallager. *Low-Density Parity-Check Codes*. PhD thesis, M.I.T., 1963.

[17] 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.

[18] 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`.

[19] 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.

[20] 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.

[21] Jingwei Hu, Wei Guo, Jizeng Wei, and Ray CC Cheung. Fast and generic inversion architectures over $gf(2^m)$ using modified itoh–tsujii algorithms. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(4):367–371, 2015.

[22] Jingwei Hu, Wen Wang, Ray CC Cheung, and Huaxiong Wang. Optimized polynomial multiplier over commutative rings on fpgas: A case study on bike. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pages 231–234. IEEE, 2019.

[23] Toshiya Itoh and Shigeo Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in GF (2m) Using Normal Bases. *Information and computation*, 78(3):171–177, 1988.

[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, Istambul, 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] 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.

[34] Nicolas Sendrier. Decoding one out of many. In B.-Y. Yang, editor, *PQCrypto 2011*, volume 7071 of *LNCS*, pages 51–67. Springer, 2011.

[35] Nicolas Sendrier and Valentin Vasseur. Backflip: An improved qc-mdpc bit-flipping decoder. CBC 2019, 2019. `https://cbc2019.dii.univpm.it`.

[36] 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, Chongquing, China, May 2019. Springer.

[37] 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.

[38] Nicolas Sendrier and Valentin Vasseur. On the existence of weak keys for QC-MDPC decoding. Cryptology ePrint Archive, Report 2020/1232, 2020.

[39] 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.

[40] 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.

[41] 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.

[42] 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* $\mathcal{C} = \{c \in \mathbb{F}_2^n \mid cH^\intercal = 0\}$*.*

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

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. Formally, it is defined as follows.

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* $n_0 r$ *and dimension* $k_0 r$*.*

A.1.2 Circulant Matrices as a Polynomial Ring

There exists a natural ring isomorphism, denoted by φ, 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 $(a_0, \ldots, a_{r-1})$ is mapped to the polynomial $\varphi(A) = a_0 + a_1 X + \cdots + a_{r-1} X^{r-1}$. This allows to view all matrix operations as polynomial operations. For every $a = a_0 + a_1 X + a_2 X^2 + \cdots + a_{r-1} X^{r-1}$ in $\mathcal{R}$, define $a^\intercal = a_0 + a_{r-1} X + \cdots + a_1 X^{r-1}$. This ensures $\varphi(A^\intercal) = \varphi(A)^\intercal$.

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

Factors of $X^r - 1$. If r is even the scheme is subject to the squaring attack [25]. If r is divisible by 2^ℓ, the attack can be repeated ℓ 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 $X^r - 1$ has no other factor than $X - 1$ in $\mathbb{F}_2[X]$ in order to eliminate 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 φ. 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 1×2 matrix (g_0, g_1) over $\mathcal{R}$ where g_0 and g_1 are the images of G_0 and G_1 by φ (*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 [16], 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 $n = n_0 r$, dimension $k = k_0 r$, 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 [16] 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

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

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 [16] and in particular with the (hard decision) bit flipping algorithm. Using floating point soft decision decoding would improve the decoding performance [3], 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 [19]. Bit flipping decoding for QC-MDPC was suggested with the original design [29]. The decoding was later improved by a better threshold selection [7, 8], 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 [35, 37], 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 [12]. 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 [12] discussed the importance of defining a decoder as an algorithm that runs a fixed number of steps (rather then 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, [12] 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 [14] 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 [14] as the most efficient variant, at least for the studied platforms (see Algorithm 1 in [14]). BIKE uses the BGF decoder with tuned threshold functions that are based on fresh extensive simulations.

Threshold Selection Rule $\texttt{threshold}(S, i)$**.** 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. The current rule do not depend of 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, an 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 [40, 36], predict a concave shape for the curve in the relevant range of r values. Assuming a similar behavior, as described in [37], 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 [12] and [14] 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 [5].

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* $eH^\intercal = s$.

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* $cH^\intercal = 0$*.*

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 $\text{p}(t) \in \{\text{odd}, \text{even}\}$.

Problem 3 ($(2, 1)$-QC Syndrome Decoding – $(2, 1)$-QCSD)**.**
Instance: $(h, s) \in \mathcal{R}_{\text{odd}} \times \mathcal{R}_{\text{p}(t)}$*, an integer* $t > 0$*.*
Property: *There exists* $(e_0, e_1) \in \mathcal{E}_t$ *such that* $e_0 + e_1 h = s$*.*

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* $h_1 + h_0 h = 0$*.*

The problems will be referred to respectively as $\text{QCSD}_{r,t}$ and $\text{QCCF}_{r,w}$, 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 $\text{QCSD}(e, h, s)$ 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 $e_0 + e_1 h = s$. For convenience, for any other input, including when the the elements are out of range (*e.g.* $e = \perp$), the value of $\text{QCSD}(e, h, s)$ is false. Similarly, the expression $\text{QCCF}(h_0, h_1, h)$ 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

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

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

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

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

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

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

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

Concretely, the hardness of Problems 3 and 4 is expressed by the fact that for all known adversaries of advantage Adv, for any of the above definitions, and running in time T, the quantity T/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 [4] 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

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

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 [41] 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 [18] give $\mathrm{WF}_{\mathrm{BJMM}}(65542, 32771, 264) = 2^{263.3}$ "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) [34] 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: $\mathrm{WF}(\mathrm{QCSD}_{r,t}) \approx \dfrac{\mathrm{WF}_{\mathrm{ISD}}(2r, r, t)}{\sqrt{r}}$
- BIKE Key Security: $\mathrm{WF}(\mathrm{QCCF}_{r,w}) \approx \dfrac{\mathrm{WF}_{\mathrm{ISD}}(2r, r, w)}{r}$

where $\mathrm{WF}(\mathrm{QCSD}_{r,t})$ and $\mathrm{WF}(\mathrm{QCCF}_{r,w})$ denote the average cost for finding a witness respectively to Problem 3 and Problem 4, and $\mathrm{WF}_{\mathrm{ISD}}$ 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 α 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 [6]). 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[42], 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 αI^2. In our setting, we can perform $I = \frac{\mathsf{MAXDEPTH}}{D_f}$ sequentially before measuring, and each iteration costs time C_f. At each iteration, we succeed with probability αI^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:

$$Q = \frac{1}{\alpha I^2} \cdot I \cdot C_f = \frac{D_f \cdot C_f}{\alpha \mathsf{MAXDEPTH}}. \tag{1}$$

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

$$Q_{AES-N} = \frac{2^N D_{AES-N} \cdot C_{AES-N}}{\mathsf{MAXDEPTH}}. \tag{2}$$

The proposition follows by comparing (1) with (2). □

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* [17] 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 not possible to completely exclude the possibility that either particular codes (weak keys, as mentioned in [12]) or particular error patterns (near-codewords, leading to error floors, as mentioned in [37]) 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 $\mathrm{DFR}(\mathcal{W})$ its relative DFR, taken on average over all errors and all keys in $\mathcal{W}$. If a set of keys is such that

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

then the (average) DFR would also be above the security requirement, even if the extrapolated failure rate was small enough. Such a set was weak keys was suggested in [12], with a relative DFR which was considerably higher than for a typical key. Later it was proven in [38] 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 [15]. Moreover, [15] 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 $\mathtt{PKE}_0$. The OW-CPA games for $\mathtt{PKE}_0$ 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 $\textsc{qcsd}(e, h, s)$ is true if and only if e is a witness of QCSD (Problem 3) for the instance (h, s). The

$\mathbf{KeyGen}_0$	Output: $(h_0, h_1) \in \mathcal{H}_w$, $h \in \mathcal{R}$ $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$; $h \leftarrow h_1 h_0^{-1}$
$\mathbf{Encrypt}_0$	Input: $h \in \mathcal{R}$, $(e_0, e_1) \in \mathcal{E}_t$ Output: $s \in \mathcal{R}$ $s \leftarrow e_0 + e_1 h$
$\mathbf{Decrypt}_0$	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: $\mathtt{PKE}_0$: 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: **return** $\text{QCSD}(e, h, s^*)$	6: **return** $\text{QCSD}(e, h, s^*)$	6: **return** $\text{QCSD}(e, h, s^*)$

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

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 $\mathtt{PKE}_0$ there exists a distinguisher D against* QCCF*, running in about the same time, such that*

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

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

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

and thus

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

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

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

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

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

□

C.1.2 From OW-CPA to IND-CPA

Table 12 describes a new encryption scheme PKE, which is essentially a randomized version of $\mathtt{PKE}_0$. It is constructed as a hybrid encryption scheme [39, 9], where the KEM part is derived from $\mathtt{PKE}_0$ as described in [10], and the DEM component is simply a one-time pad. Unlike $\mathtt{PKE}_0$, 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 PKE is IND-CPA secure, reducing tightly to the OW-CPA security of $\mathtt{PKE}_0$. The hash function $\mathbf{L}$ is modeled as random oracle for purpose of the proof, which is inspired again to [10].

$\mathbf{KeyGen}_0$	Output: $(h_0, h_1) \in \mathcal{H}_w$, $h \in \mathcal{R}$ $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$; $h \leftarrow h_1 h_0^{-1}$
Encrypt	Input: $h \in \mathcal{R}$, $m \in \mathcal{M}$ Output: $c \in \mathcal{R} \times \mathcal{M}$ $(e_0, e_1) \xleftarrow{\$} \mathcal{E}_t$; $c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$
Decrypt	Input: $(h_0, h_1) \in \mathcal{H}_w$, (c_0, c_1) Output: $m \in \mathcal{M} \cup \{\perp\}$ $e \leftarrow \text{decoder}(sh_0, h_0, h_1)$ **if** $e = \perp$ **then** $m \leftarrow \perp$ **else** $m \leftarrow c_1 \oplus \mathbf{L}(e)$

Table 12: PKE: Randomization of $\mathtt{PKE}_0$

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

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

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

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

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

Game G_0 (IND-CPA)	**Game** $G_1{}^{(\dagger)}$	**Game** $G_2{}^{(\dagger)}$
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: $c_1^* \xleftarrow{\$} m_b^* \oplus \mathbf{L}(e^*)$	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: **return** $b = b'$?	9: **return** $b = b'$?	9: **return** $b = b'$?

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

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

denotes the set of witnesses of QCSD 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

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

2. 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

Game G_3'	$\mathbf{L}'(e)$:	**Game** G_3 (OW-CPA)
1: $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$	**if** $\text{QCSD}(e, h, c_0^*)$ **then**	1: $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$
2: $h \leftarrow h_1 h_0^{-1}$	$\text{WT} \leftarrow e$	2: $h \leftarrow h_1 h_0^{-1}$
3: $e^* = (e_0^*, e_1^*) \xleftarrow{\$} \mathcal{E}_t$	**return** $\mathbf{L}(e)$	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: $b \xleftarrow{\$} \{0,1\}$; $\text{WT} \leftarrow \perp$	5: $b \xleftarrow{\$} \{0,1\}$; $\text{WT} \leftarrow \perp$	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: **return** $\text{QCSD}(e, h, s^*)$
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: **return** $\text{QCSD}(\text{WT}, h, c_0^*)$	9: **return** WT	

Table 14: Adversary for $\texttt{PKE}_0$

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 $\texttt{PKE}_0$, 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:

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

□

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*

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

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 [20], $\mathtt{PKE}$ is defined to be δ-correct if

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

where the expectation is taken over $(pk, sk) \leftarrow \mathbf{KeyGen}^{\mathtt{PKE}}$. Failure to decrypt a ciphertext ($c_0 = e_0 + e_1 h, c_1$) happens if and only if $(e_0, e_1) \neq \mathtt{decoder}(c_0 h_0, h_0, h_1)$ and c_0 only depends on the internal randomness, not on the message m. This property is referred to as *message-agnostic* in [15]. The max vanishes in (3). It follows that $\mathtt{PKE}$ is δ-correct for any δ such that

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

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

C.2.2 HHK Proof

$\mathbf{KeyGen}_0$	Output: $(h_0, h_1) \in \mathcal{H}_w$, $h \in \mathcal{R}$ $(h_0, h_1) \xleftarrow{\$} \mathcal{H}_w$; $h \leftarrow h_1 h_0^{-1}$
$\mathbf{Encrypt}_1$	Input: $h \in \mathcal{R}$, $m \in \mathcal{M}$ Output: $c \in \mathcal{R} \times \mathcal{M}$ $(e_0, e_1) \leftarrow \mathbf{H}(m)$; $c \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$
$\mathbf{Decrypt}_1$	Input: $(h_0, h_1) \in \mathcal{H}_w$, $c \in \mathcal{R} \times \mathcal{M}$, $h \in \mathcal{R}$ Output: $m \in \mathcal{M} \cup \{\perp\}$ $m \leftarrow \mathbf{Decrypt}((h_0, h_1), c)$ **if** $m \neq \perp$ **and** $c \neq \mathbf{Encrypt}_1(h, m)$ **then** $m \leftarrow \perp$

Table 15: $\mathtt{PKE}_1$: Derandomizing $\mathtt{PKE}$

The proof framework of [20] transforms a probabilistic public-key encryption scheme, here $\mathtt{PKE}$ (Table 12), first into a derandomized variant $\mathtt{PKE}_1$ (Table 15) then into a key encapsulation mechanism with implicit rejection $\mathtt{KEM}^{\not\perp}$ (Table 16). A hash function $\mathbf{H}$ is required for $\mathtt{PKE}_1$ and another one $\mathbf{K}$ for $\mathtt{KEM}^{\not\perp}$.

Lemma 2. *[20, §3.3] If* $\mathtt{PKE}$ *is* δ*-correct, for all IND-CCA adversary* B *against* $\mathtt{KEM}^{\not\perp}$ *issuing at most* q *queries to* $\mathbf{K}$ *or* $\mathbf{H}$*, there exists an IND-CPA adversary* A *against* $\mathtt{PKE}$*, running in about the same time, such that*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\mathtt{KEM}^{\not\perp}}(B) \leq q \cdot \delta + \frac{3 \cdot q}{|\mathcal{M}|} + 3 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathtt{PKE}}(A).$$

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

Table 16: $\texttt{KEM}^{\not\perp}$: KEM with Implicit Rejection From $\texttt{PKE}_1$

Proof (sketch). There are two key theorems in [20] to prove that $\texttt{KEM}^{\not\perp}$ is IND-CCA secure. One relates the OW-PCVA security of $\texttt{PKE}_1$ to the IND-CPA security of PKE. The other relates the OW-PCA security of $\texttt{PKE}_1$ with the IND-CCA security of $\texttt{KEM}^{\not\perp}$. [20, Theorem 3.4] states that for all IND-CCA adversaries B against $\texttt{KEM}^{\not\perp}$, issuing at most q_K queries to $\mathbf{K}$, there exists an OW-PCA adversary B' against $\texttt{PKE}_1$, running in about the same time, issuing at most q_K queries to Pco such that

$$\mathsf{Adv}^{\text{IND-CCA}}_{\texttt{KEM}^{\not\perp}}(B) \leq \frac{q_K}{|\mathcal{M}|} + \mathsf{Adv}^{\text{OW-PCA}}_{\texttt{PKE}_1}(B')$$

Pco is a plaintext checking oracle which returns true on input (m, c) if and only if $m = \mathbf{Decrypt}_1((h_0, h_1), c)$. [20, 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 δ-correct, [20, Theorem 3.2] states that for all OW-PCA adversary B' against $\texttt{PKE}_1$ 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

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

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. □

C.3 The BIKE Key Encapsulation Mechanism

The construction proposed in [20] is convenient because it provides easy tools for the proof. To obtain a self-contained description of $\texttt{KEM}^{\not\perp}$, 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.

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

Table 17: $\texttt{KEM}^{\not\perp}$ 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 δ-correct, for all IND-CCA adversary B against* $\texttt{KEM}^{\not\perp}$ *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*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\texttt{KEM}^{\not\perp}}(B) \leq q \cdot \delta + \frac{3 \cdot q}{|\mathcal{M}|} + \frac{3}{2}\mathsf{Adv}^{\text{IND}}_{\text{QCCF}}(D) + \frac{3}{2}\mathsf{Adv}^{\text{OW}}_{\text{QCSD}}(A'). \tag{4}$$

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

C.3.1 Concrete Security and Parameters Selection

To offer λ bits of security it is typically required that $|A| / \mathsf{Adv}(A) \geq 2^\lambda$ 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 $\texttt{KEM}^{\not\perp}$, the BIKE key encapsulation mechanism, offers λ bits of (classical) security in the IND-CCA game if the system parameters r, w, t, ℓ, and `decoder` are selected at setup such that

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

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 ℓ 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 $\mathcal{R}$.