BIKE:
Bit Flipping Key Encapsulation

Round 2 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, INSA-CVL Bourges and University of Limoges, France

Philippe Gaborit, University of Limoges, France

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, Intel Corporation, 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): rafael.misoczki@intel.com

Postal Address and Telephone (if absolutely necessary):
Rafael Misoczki, Intel Corporation, Jones Farm 2 Building, 2111 NE 25th
Avenue, Hillsboro, OR 97124, +1 (503) 264 0392.

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

Version: 3.0

Release Date: March 30th, 2019

Contents

1 Introduction

This document describes BIKE, a suite of algorithms for key encapsulation based on quasi-cyclic moderate density parity-check (QC-MDPC) codes that can be decoded using bit flipping decoding techniques. In particular, this document highlights the number of security, performance and simplicity advantages that make BIKE a compelling candidate for post-quantum key encapsulation standardization.

1.1 Notation and Preliminaries

Table 1 presents the used notation and is followed by preliminary concepts.

Notation	Description
$\mathbb{F}_2$:	Finite field of 2 elements.
$\mathcal{R}$:	The cyclic polynomial ring $\mathbb{F}_2[X]/\langle X^r - 1\rangle$.
$\lvert v\rvert$:	The Hamming weight of a binary polynomial v.
$u \xleftarrow{\$} U$:	Variable u is sampled uniformly at random from set U.
h_j:	The j-th column of a matrix H, as a row vector.
$\star$:	The component-wise product of vectors.

Table 1: Notation

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* $\mathcal{C}$ *iff*

$$\mathcal{C} = \{mG \mid m \in \mathbb{F}_2^k\}.$$

A matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ *is called a* parity-check matrix *of* $\mathcal{C}$ *iff*

$$\mathcal{C} = \{c \in \mathbb{F}_2^n \mid Hc^T = 0\}.$$

A *codeword* $c \in \mathcal{C}$ of a vector $m \in \mathbb{F}_2^{(n-r)}$ is computed as $c = mG$. A *syndrome* $s \in \mathbb{F}_2^r$ of a vector $e \in \mathbb{F}_2^n$ is computed as $s^T = He^T$.

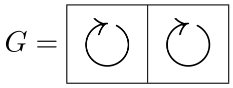
1.2 Quasi-Cyclic Codes

A binary circulant matrix is a square matrix where each row is the rotation 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.

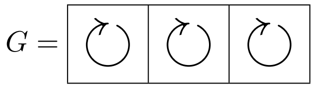
1.2.1 Definition

Definition 3 (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 $k_0 r$.*

For instance:



The rows of G span a $(2, 1)$-QC code



The rows of G span a $(3, 1)$-QC code

1.2.2 Representation of QC Codes

Representation of Circulant Matrices. There exists a natural ring isomorphism, which we denote 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 will allow us to view all matrix operations as polynomial operations.

Transposition. For any $a = a_0 + a_1 X + a_2 X^2 + \cdots + a_{r-1} X^{r-1}$ in $\mathcal{R}$, we define $a^T = a_0 + a_{r-1} X + \cdots + a_1 X^{r-1}$. This will ensure $\varphi(A^T) = \varphi(A)^T$.

Vector/Matrix Product. We may extend the mapping φ to any binary vector of $\mathbb{F}_2^r$. For all $\mathbf{v} = (v_0, v_1, \ldots, v_{r-1})$, we set $\varphi(\mathbf{v}) = v_0 + v_1 X + \cdots + v_{r-1} X^{r-1}$. To stay consistent with the transposition, the image of the column vector $\mathbf{v}^T$ must be $\varphi(\mathbf{v}^T) = \varphi(\mathbf{v})^T = v_0 + v_{r-1} X + \cdots + v_1 X^{r-1}$. It is easily checked that $\varphi(\mathbf{v}A) = \varphi(\mathbf{v})\varphi(A)$ and $\varphi(A\mathbf{v}^T) = \varphi(A)\varphi(\mathbf{v})^T$.

Representation of QC Codes as Codes over a Polynomial Ring. The generator matrix of an (n_0, k_0)-QC code can be represented as an $k_0 \times n_0$ matrix over $\mathcal{R}$. 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 $\mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$.

1.3 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 $O(1/\sqrt{n})$. The existence of such a matrix allows the use of iterative decoders similar to those used for LDPC (Low Density Parity Check) codes [17], widely deployed for error correction in telecommunication.

1.3.1 Definition

Definition 4 (QC-MDPC codes)**.** *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 $w = O(\sqrt{n})$.*

Remark 1. *Asymptotically, a QC-MDPC code could efficiently correct up to $t = O(\sqrt{n} \log n)$ errors. This is a corollary of Theorem 1 given in paragraph "Asymptotic Analysis for MDPC Codes" that follows. In this work, the parity-check row weight w and the error weight t will be chosen so that $wt = O(n)$. This is precisely the regime where the decryption failure rate is expected to decay exponentially in the codelength n (see Theorem 1).*

1.3.2 Decoding - The Bit Flipping Algorithm

The decoding of MDPC codes can be achieved by various iterative decoders. Among those, the *bit flipping algorithm* is particularly interesting because of its simplicity. In Algorithm 1 as it is given here the instruction to determine the threshold τ is unspecified. We will always consider regular codes, where all columns of h have the same weight d and we denote $T = \tau d$. There are several rules for computing the threshold T:

Algorithm 1 `Bit Flipping Algorithm`

```
Require: $H \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$
Ensure: $eH^T = s$
1: $e \leftarrow 0$
2: $s' \leftarrow s$
3: while $s' \neq 0$ do
4:     $\tau \leftarrow$ threshold $\in [0,1]$, found according to some predefined rule
5:     for $j = 0, \dots, n-1$ do
6:         if $|h_j \star s'| \geq \tau \, |h_j|$ then
7:             $e_j \leftarrow e_j + 1 \mod 2$
8:     $s' \leftarrow s - eH^T$
9: return $e$
```

h_j denotes the j-th column of H, as a row vector, $'\star'$ denotes the component-wise product of vectors, and $|h_j \star s|$ is the number of unchecked parity equations involving j.

- the maximal value of $|h_j \star s|$ minus some δ (typically $\delta = 5$), as in [36],
- precomputed values depending on the iteration depth, as in [12],
- variable, depending on the weight of the syndrome s', as in [11].

The algorithm takes as input a parity check matrix H and a word s and, if it stops, returns an error pattern e whose syndrome is s. If H is sparse enough and there exists an error e of small enough weight such that $s = eH^T$, then, with high probability, the algorithm stops and returns e.

Asymptotic Analysis for MDPC Codes. For a fixed code rate k/n, let us denote w the weight of the rows of H and t the number of errors we are able to decode. Both w and t are functions of n. For LDPC codes, w is a constant and t will be a constant proportion of n, that is $wt = \Omega(n)$. For MDPC codes, we have $w = \Omega(\sqrt{n})$ and the amount of correctable errors will turn out to be a little bit higher than $t = \Omega(\sqrt{n})$.

To understand this point, let us first notice that experimental evidence seems to indicate that the decryption failure rate is dominated by the probability that the first round of the algorithm is unable to reduce significantly the number of initial errors. What we call here "round" of the decoding algorithm is an execution of the for-loop of line 5 of Algorithm 1. It also seems that at the first round of the decoding algorithm the individual bits of the syndrome bits s_i can be approximated by independent random variables. This independence assumption can also be made for the vectors $h_j \star s = h_j \star s'$ at the first round. In other words, we make the following assumptions.

Assumption 1. *Let P_{err} be the probability that the bit flipping algorithm fails to decode. Let e^1 be the value of error-vector e after executing the for-loop of line 5 of Algorithm 1 and let e^0 be the true error vector. Let $\Delta e = e^0 + e^1$ (addition is performed in $\mathbb{F}_2$) be the error vector that would remain if we applied the correction e^1 to the true error vector e^0.*

- *There exists a constant α in $(0,1)$ such that*
$$P_{err} \leq \mathbb{P}(|\Delta e| \geq \alpha t).$$
- *The syndrome bits s_i are independent random variables.*
- *For $j = 0, \dots, n-1$, the $h_j \star s$ are independent random variables.*

By making these assumptions we can prove that

Theorem 1. *Under assumption 1, the probability P_{err} that the bit flipping algorithm fails to decode with fixed threshold $\tau = \frac{1}{2}$ is upper-bounded by*

$$P_{err} \leq \frac{1}{\sqrt{\alpha \pi t}} e^{\frac{\alpha t w}{8} \ln\left(1-\varepsilon^2\right) + \frac{\alpha t}{8} \ln(n) + O(t)},$$

where $\varepsilon \overset{def}{=} e^{-\frac{2wt}{n}}$.

This theorem is proved in Appendix A. This theorem shows that the decryption failure rate (DFR) decays exponentially in the codelength when $wt = O(n)$ and that the number of correctable errors is a little bit larger than $O(\sqrt{n})$ when $w = O(\sqrt{n})$: it can be as large as some constant $\beta\sqrt{n}\ln n$ as the upper-bound in this theorem is easily shown to converge to 0 for a small enough constant β.

Decoding with a Noisy Syndrome. Noisy syndrome decoding is a variation of syndrome decoding in which, given H and s, we look for $e \in \mathbb{F}_2^n$ such that $s - eH^T$ and e are both of small weight. The bit flipping algorithm can be adapted to noisy syndromes. Two things must be modified. First the stopping condition: we do not require the quantity $s - eH^T$ to be null, only to have a small weight. Second, since we need to quantify the weight in this stopping condition, we need to specify a target weight u. For input (H, s, u) a pair e is returned such that $s = e' + eH^T$ for some e' of weight at most u. If $u = 0$ we have the usual bit flipping algorithm.

Algorithm 2 `Extended Bit Flipping Algorithm`

```
Require: $H \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$, integer $u \geq 0$
Ensure: $\left|s - eH^T\right| \leq u$
$e \leftarrow 0$
$s' \leftarrow s$
while $|s'| > u$ do
    $\tau \leftarrow$ threshold $\in [0,1]$, found according to some predefined rule
    for $j = 0, \ldots, n-1$ do
        if $|h_j \star s'| \geq \tau |h_j|$ then
            $e_j \leftarrow e_j + 1 \mod 2$
    $s' \leftarrow s - eH^T$
return $e$
```

Again, if H is sparse enough and there exists a solution, the algorithm will stop with high probability. Note that if the algorithm stops, it returns a solution within the prescribed weight, but this solution might not be unique. In the case of MDPC codes, the column weight and the error weight are both of order $\sqrt{r}$ and the solution is unique with high probability.

Noisy Syndrome *vs.* Normal Bit Flipping. Interestingly, for MDPC codes, noisy syndromes affect only marginally the performance of the bit flipping algorithm. In fact, if e is the solution of $s = e' + eH^T$, then it is also the solution of $s = (e, 1)H'^T$ where H' is obtained by appending e' as $n+1$-th column. For MDPC codes, the error vector e' has a density which is similar to that of H and thus H' is sparse and its last column is not remarkably more or less sparse. Thus applying the bit flipping algorithm to (H', s) is going to produce e, except that we do not allow the last position to be tested in the loop and control is modified to stop the loop when the syndrome s' is equal to the last column of H'. Since we never test the last position we don't need to know the value of the last column of H' except for the stopping condition which can be replaced by a test on the weight. Thus we emulate (almost) the noisy syndrome bit flipping by running the bit flipping algorithm on a code of length $n+1$ instead of n, to correct $|e|+1$ errors instead of $|e|$.

QC-MDPC Decoding for Decryption. Quasi-cyclicity does not change the decoding algorithm. The above algorithm will be used for $(2, 1)$-QC MDPC codes. It allows us to define the procedure specified as follows. For any triple $(s, h_0, h_1) \in \mathcal{R}^3$ and any integer u

$$\texttt{Decode}(s, h_0, h_1, u) \text{ returns } (e_0, e_1) \in \mathcal{R}^2 \text{ with } |e_0h_0 + e_1h_1 + s| \leq u.$$

The fourth argument u is an integer. If $u = 0$, the algorithm stops when $e_0h_0 + e_1h_1 = s$, that is the noiseless syndrome decoding, otherwise it stops when $e_0h_0 + e_1h_1 = s+e$ from some e of weight at most u, that is the noisy syndrome decoding. In addition, we will bound the running time (as a function of the block size r) and stop with a failure when this bound is exceeded.

1.4 Key Encapsulation Mechanisms

A key encapsulation mechanism (KEM) is composed by three algorithms: Gen which outputs a public encapsulation key *pk* and a private decapsulation key *sk*, Encaps which takes as input an encapsulation key *pk* and outputs a ciphertext c and a symmetric key K, and Decaps which takes as input a decapsulation key *sk* and a cryptogram c and outputs a symmetric key K or a decapsulation failure symbol $\perp$. For more details on KEM definitions, we refer the reader to [14].

2 Algorithm Specification (2.B.1)

2.1 IND-CPA Variants

In this section, we decribe the BIKE variants that achieve IND-CPA security. These variants use ephemeral keys, meaning that a new key pair is generated at each key exchange. In this way, forward security is achieved. Additionally, attack strategies that depend on the observation of a large number of decoding failures for a same private key, such as [22], are not applicable.

In the following we will present three IND-CPA secure variants of BIKE, which we will simply label BIKE-1, BIKE-2 and BIKE-3. All of the variants follow either the McEliece or the Niederreiter framework, but each one has some important differences, which we will discuss individually.

For a security level λ, let r be a prime such that $(X^r - 1)/(X - 1) \in \mathbb{F}_2[X]$ is irreducible, d_v be an odd integer and t be an integer such that decoding t errors with a uniformly chosen binary linear error-correcting code of length $n = 2r$ and dimension r, as well as recovering a base of column weight d_v given an arbitrary base of a code of the same length and dimension, both have a computational cost in $\Omega(\exp(\lambda))$. See Section 5 for a detailed discussion on parameters selection.

We denote by $\mathbf{K} : \{0,1\}^n \rightarrow \{0,1\}^{\ell_K}$ the hash function used by encapsulation and decapsulation, where ℓ_K is the desired symmetric key length (typically 256 bits).

2.1.1 BIKE-1

In this variant, we privilege a fast key generation by using a variation of McEliece. A preliminary version of this approach appears in [4].

First, in contrast to QC-MDPC McEliece [36] (and any QC McEliece variant), we do not compute the inversion of one of the private cyclic blocks and then multiply it by the whole private matrix to get systematic form. Instead, we hide the private code structure by simply multiplying its sparse private matrix by any random, dense cyclic block. The price to pay is the doubled size for the public key and the data since the public key will not feature an identity block anymore.

Secondly, we interpret McEliece encryption as having the message conveyed in the error vector, rather than the codeword. This technique is not new, following the lines of Micciancio's work in [35] and having already been used in a code-based scheme by Cayrel et al. in [9].

KeyGen

- Input: λ, the target quantum security level.
- Output: the sparse private key (h_0, h_1) and the dense public key (f_0, f_1).

0. Given λ, set the parameters r, w as described above.
1. Generate $h_0, h_1 \xleftarrow{\$} \mathcal{R}$ both of (odd) weight $|h_0| = |h_1| = w/2$.
2. Generate $g \xleftarrow{\$} \mathcal{R}$ of odd weight (so $|g| \approx r/2$).
3. Compute $(f_0, f_1) \leftarrow (gh_1, gh_0)$.

Encaps

- Input: the dense public key (f_0, f_1).
- Output: the encapsulated key K and the cryptogram c.

1. Sample $(e_0, e_1) \in \mathcal{R}^2$ such that $|e_0| + |e_1| = t$.
2. Generate $m \xleftarrow{\$} \mathcal{R}$.
3. Compute $c = (c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$.
4. Compute $K \leftarrow \mathbf{K}(e_0, e_1)$.

Decaps

- Input: the sparse private key (h_0, h_1) and the cryptogram c.
- Output: the decapsulated key K or a failure symbol $\perp$.

1. Compute the syndrome $s \leftarrow c_0 h_0 + c_1 h_1$.
2. Try to decode s (noiseless) to recover an error vector (e'_0, e'_1).
3. If $|(e'_0, e'_1)| \neq t$ or decoding fails, output $\perp$ and halt.
4. Compute $K \leftarrow \mathbf{K}(e'_0, e'_1)$.

2.1.2 BIKE-2

In this variant, we follow Niederreiter's framework with a systematic parity check matrix. The main advantage is that this only requires a single block of length r for all the objects involved in the scheme, and thus yields a very compact formulation. On the other hand, this means that it is necessary to perform a polynomial inversion. In this regard, it is worth mentioning that an inversion-based key generation can be significantly slower than encryption (e.g., up to 21x as reported in [33]). A possible solution is to use a batch key generation as described in Section 3.7.

KeyGen

- Input: λ, the target quantum security level.
- Output: the sparse private key (h_0, h_1) and the dense public key h.

0. Given λ, set the parameters r, w as described above.
1. Generate $h_0, h_1 \xleftarrow{\$} \mathcal{R}$ both of (odd) weight $|h_0| = |h_1| = w/2$.
2. Compute $h \leftarrow h_1 h_0^{-1}$.

Encaps

- Input: the dense public key h.
- Output: the encapsulated key K and the cryptogram c.

1. Sample $(e_0, e_1) \in \mathcal{R}^2$ such that $|e_0| + |e_1| = t$.
2. Compute $c \leftarrow e_0 + e_1 h$.
3. Compute $K \leftarrow \mathbf{K}(e_0, e_1)$.

Decaps

- Input: the sparse private key (h_0, h_1) and the cryptogram c.
- Output: the decapsulated key K or a failure symbol $\perp$.

1. Compute the syndrome $s \leftarrow ch_0$.
2. Try to decode s (noiseless) to recover an error vector (e'_0, e'_1).
3. If $|(e'_0, e'_1)| \neq t$ or decoding fails, output $\perp$ and halt.
4. Compute $K \leftarrow \mathbf{K}(e'_0, e'_1)$.

2.1.3 BIKE-3

This variant follows the work of Ouroboros [15]. Looking at the algorithms description, the variant resembles BIKE-1, featuring fast, inversion-less key generation and two blocks for public key and data. The main difference is that the decapsulation invokes the decoding algorithm on a "noisy" syndrome. This also means that BIKE-3 is fundamentally distinct from BIKE-1 and BIKE-2, mainly in terms of security and security-related aspects like choice of parameters. We will discuss this in the appropriate section.

KeyGen

- Input: λ, the target quantum security level.
- Output: the sparse private key (h_0, h_1) and the dense public key (f_0, f_1).

0. Given λ, set the parameters r, w as described above.
1. Generate $h_0, h_1 \xleftarrow{\$} \mathcal{R}$ both of (odd) weight $|h_0| = |h_1| = w/2$.
2. Generate $g \xleftarrow{\$} \mathcal{R}$ of odd weight (so $|g| \approx r/2$).
3. Compute $(f_0, f_1) \leftarrow (h_1 + gh_0, g)$.

Encaps

- Input: the dense public key (f_0, f_1).
- Output: the encapsulated key K and the cryptogram c.

1. Sample $(e, e_0, e_1) \in \mathcal{R}^3$ with $|e| = t/2$ and $|e_0| + |e_1| = t$.
2. Compute $c = (c_0, c_1) \leftarrow (e + e_1 f_0, e_0 + e_1 f_1)$.
3. Compute $K \leftarrow \mathbf{K}(e_0, e_1)$.

Decaps

- Input: the sparse private key (h_0, h_1) and the cryptogram c.
- Output: the decapsulated key K or a failure symbol $\perp$.

1. Compute the syndrome $s \leftarrow c_0 + c_1 h_0$.
2. Try to decode s (with noise at most $t/2$) to recover error vector (e'_0, e'_1).
3. If $|(e'_0, e'_1)| \neq t$ or decoding fails, output $\perp$ and halt.
4. Compute $K \leftarrow \mathbf{K}(e'_0, e'_1)$.

Comparison between IND-CPA BIKE variants. For ease of comparison, we provide a summary of the three schemes in Table 2 below.

	BIKE-1	BIKE-2	BIKE-3
SK	(h_0, h_1) with $\vert h_0\vert = \vert h_1\vert = w/2$		
PK	$(f_0, f_1) \leftarrow (gh_1, gh_0)$	$(f_0, f_1) \leftarrow (1, h_1 h_0^{-1})$	$(f_0, f_1) \leftarrow (h_1 + gh_0, g)$
Enc	$(c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$	$c \leftarrow e_0 + e_1 f_1$	$(c_0, c_1) \leftarrow (e + e_1 f_0, e_0 + e_1 f_1)$
	$K \leftarrow \mathbf{K}(e_0, e_1)$		
Dec	$s \leftarrow c_0 h_0 + c_1 h_1$; $u \leftarrow 0$	$s \leftarrow ch_0$; $u \leftarrow 0$	$s \leftarrow c_0 + c_1 h_0$; $u \leftarrow t/2$
	$(e'_0, e'_1) \leftarrow \texttt{Decode}(s, h_0, h_1, u)$		
	$K \leftarrow \mathbf{K}(e'_0, e'_1)$		

Table 2: Algorithm Comparison

We remark that e can be represented with only $\lceil \log_2 \binom{n}{t} \rceil$ bits and such a compact representation can be used if memory is the preferred metric of optimization (the hash function $\mathbf{K}$ would need to be changed as well to receive $\lceil \log_2 \binom{n}{t} \rceil$ bits instead of n).

2.2 IND-CCA Variants

This version of BIKE is designed to make use of static keys, meaning that several key exchanges can take place with the same key pair. This is possible thanks to the improved Backflip decoder, which yields an extremely small number of decoding failures, corresponding of the desired security level. Moreover, a small decoding failure rate inhibits the GJS attack.

In the following we will present three IND-CCA secure variants of BIKE, called BIKE-1-CCA, BIKE-2-CCA and BIKE-3-CCA, corresponding to their respective

IND-CPA counterparts. All of the variants are obtained by applying a specific conversion to the underlying cryptosystem, and we will discuss the details of these conversions individually in Section 6.2.

Parameters are chosen as before, with one exception. In fact, in the case of BIKE-1-CCA and BIKE-3-CCA, the conversion requires to use a random oracle to transform the encryption scheme from probabilistic to deterministic. Therefore, in addition to the hash function $\mathbf{K} : \{0,1\}^{2n} \to \{0,1\}^{\ell_K}$ used to derive a shared key, we introduce two hash functions $\mathbf{G} : \{0,1\}^n \to \{0,1\}^r$ and $\bar{\mathbf{G}} : \{0,1\}^n \to \{0,1\}^r$, with the task of generating the randomness for the scheme. Note that, as we will see later, the range of $\bar{\mathbf{G}}$ is a subset of $\{0,1\}^r$, and the function returns elements of weight $t/2$.

2.2.1 BIKE-1-CCA

Like its IND-CPA counterpart, this variant features a very fast key generation since it avoids polynomial inversion. Moreover, as before, we interpret the underlying cryptosystem as having the message is conveyed in the error vector, and the randomness in the codeword. This is now even more relevant, since it means that this randomness will be obtained via the hash function $\mathbf{G}$.

KeyGen

- Input: λ, the target quantum security level.
- Output: the private key $(h_0, h_1, \sigma_0, \sigma_1)$ and the public key (f_0, f_1).

0. Given λ, set the parameters r, w as described above.
1. Generate $h_0, h_1 \xleftarrow{\$} \mathcal{R}$ both of (odd) weight $|h_0| = |h_1| = w/2$.
2. Generate $\sigma_0, \sigma_1 \xleftarrow{\$} \mathcal{R}$ uniformly at random.
3. Generate $g \xleftarrow{\$} \mathcal{R}$ of odd weight (so $|g| \approx r/2$).
4. Compute $(f_0, f_1) \leftarrow (gh_1, gh_0)$.

Encaps

- Input: the public key (f_0, f_1).
- Output: the encapsulated key K and the cryptogram c.

1. Sample $(e_0, e_1) \in \mathcal{R}^2$ such that $|e_0| + |e_1| = t$.
2. Generate $m \leftarrow \mathbf{G}(e_0, e_1)$.
3. Compute $c = (c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$.
4. Compute $K \leftarrow \mathbf{K}(e_0, e_1, c_0, c_1)$.

Decaps

- Input: the private key $(h_0, h_1, \sigma_0, \sigma_1)$ and the cryptogram c.
- Output: the decapsulated key K.

1. Compute the syndrome $s \leftarrow c_0 h_0 + c_1 h_1$.
2. Try to decode s (noiseless) to recover an error vector (e'_0, e'_1).
3. Generate $m' \leftarrow \mathbf{G}(e'_0, e'_1)$.
4. Compute $c' = (c'_0, c'_1) \leftarrow (m' f_0 + e'_0, m' f_1 + e'_1)$.
5. If $|(e'_0, e'_1)| \neq t$, decoding fails, or $c \neq c'$, compute $K \leftarrow \mathbf{K}(\sigma_0, \sigma_1, c'_0, c'_1)$.
6. Else compute $K \leftarrow \mathbf{K}(e'_0, e'_1, c'_0, c'_1)$.

2.2.2 BIKE-2-CCA

As before, this variant is based on the Niederreiter cryptosystem. As we will see in Section 6.2, this not only yields a very compact formulation, but also a tighter security reduction, due to the deterministic nature of the cryptosystem.

KeyGen

- Input: λ, the target quantum security level.
- Output: the private key $(h_0, h_1, \sigma_0, \sigma_1)$ and the public key h.

0. Given λ, set the parameters r, w as described above.
1. Generate $h_0, h_1 \xleftarrow{\$} \mathcal{R}$ both of (odd) weight $|h_0| = |h_1| = w/2$.
2. Generate $\sigma_0, \sigma_1 \xleftarrow{\$} \mathcal{R}$ uniformly at random.
3. Compute $h \leftarrow h_1 h_0^{-1}$.

Encaps

- Input: the public key h.
- Output: the encapsulated key K and the cryptogram c.

1. Sample $(e_0, e_1) \in \mathcal{R}^2$ such that $|e_0| + |e_1| = t$.
2. Compute $c \leftarrow e_0 + e_1 h$.
3. Compute $K \leftarrow \mathbf{K}(e_0, e_1, c)$.

Decaps

- Input: the private key $(h_0, h_1, \sigma_0, \sigma_1)$ and the cryptogram c.
- Output: the decapsulated key K.

1. Compute the syndrome $s \leftarrow ch_0$.
2. Try to decode s (noiseless) to recover an error vector (e'_0, e'_1).
3. Compute $c' \leftarrow e'_0 + e'_1 h$.
4. If $|(e'_0, e'_1)| \neq t$, decoding fails, or $c \neq c'$, compute $K \leftarrow \mathbf{K}(\sigma_0, \sigma_1, c')$.
5. Compute $K \leftarrow \mathbf{K}(e'_0, e'_1, c')$.

2.2.3 BIKE-3-CCA

As in BIKE-1-CCA, this variant features fast, inversion-less key generation and two blocks for public key and data. Also, like in its IND-CPA counterpart, the scheme uses a "noisy" version of the Backflip decoder. Finally, we once again determine the randomness of the scheme via the dedicated hash function $\bar{\mathbf{G}}$.

KeyGen

- Input: λ, the target quantum security level.
- Output: the private key $(h_0, h_1, \sigma_0, \sigma_1)$ and the public key (f_0, f_1).

0. Given λ, set the parameters r, w as described above.
1. Generate $h_0, h_1 \xleftarrow{\$} \mathcal{R}$ both of (odd) weight $|h_0| = |h_1| = w/2$.
2. Generate $\sigma_0, \sigma_1 \xleftarrow{\$} \mathcal{R}$ uniformly at random.
3. Generate $g \xleftarrow{\$} \mathcal{R}$ of odd weight (so $|g| \approx r/2$).
4. Compute $(f_0, f_1) \leftarrow (h_1 + gh_0, g)$.

Encaps

- Input: the public key (f_0, f_1).
- Output: the encapsulated key K and the cryptogram c.

1. Sample $(e_0, e_1) \in \mathcal{R}^2$ such that $|e_0| + |e_1| = t$.
2. Generate $e \leftarrow \bar{\mathbf{G}}(e_0, e_1)$ with $|e| = t/2$.
3. Compute $c = (c_0, c_1) \leftarrow (e + e_1 f_0, e_0 + e_1 f_1)$.
4. Compute $K \leftarrow \mathbf{K}(e_0, e_1, c_0, c_1)$.

Decaps

- Input: the private key $(h_0, h_1, \sigma_0, \sigma_1)$ and the cryptogram c.
- Output: the decapsulated key K.

1. Compute the syndrome $s \leftarrow c_0 + c_1 h_0$.
2. Try to decode s (with noise at most $t/2$) to recover error vector (e'_0, e'_1).
3. Generate $e' \leftarrow \bar{\mathbf{G}}(e'_0, e'_1)$.
4. Compute $c' = (c'_0, c'_1) \leftarrow (e' + e'_1 f_0, e'_0 + e'_1 f_1)$.
5. If $|(e'_0, e'_1)| \neq t$, decoding fails, or $c \neq c'$, compute $K \leftarrow \mathbf{K}(\sigma_0, \sigma_1, c'_0, c'_1)$.
6. Else compute $K \leftarrow \mathbf{K}(e'_0, e'_1, c'_0, c'_1)$.

Comparison between BIKE-CCA versions. For ease of comparison, we provide a summary of the three schemes in Table 3 below.

	BIKE-1	BIKE-2	BIKE-3
SK	$(h_0, h_1, \sigma_0, \sigma_1)$ with $\lvert h_0 \rvert = \lvert h_1 \rvert = w/2$		
PK	$(f_0, f_1) \leftarrow (gh_1, gh_0)$	$(f_0, f_1) \leftarrow (1, h_1 h_0^{-1})$	$(f_0, f_1) \leftarrow (h_1 + gh_0, g)$
Enc	$(c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$	$c \leftarrow e_0 + e_1 f_1$	$(c_0, c_1) \leftarrow (e + e_1 f_0, e_0 + e_1 f_1)$
	$K \leftarrow \mathbf{K}(e_0, e_1, c)$		
Dec	$s \leftarrow c_0 h_0 + c_1 h_1$; $u \leftarrow 0$	$s \leftarrow ch_0$; $u \leftarrow 0$	$s \leftarrow c_0 + c_1 h_0$; $u \leftarrow t/2$
	$(e'_0, e'_1) \leftarrow \texttt{Decode}(s, h_0, h_1, u)$		
	$K \leftarrow \mathbf{K}(e'_0, e'_1, c')$		

Table 3: Algorithm Comparison

Remarks about CCA Conversions There are some common traits about the conversions used to derive the various BIKE-CCA, and some aspects that change from the IND-CPA counterparts. First of all, the private key now always includes the additional random string $\sigma = (\sigma_0, \sigma_1)$ (and technically, from an implementation point of view, a copy of the public key). This is used during decapsulation in the event of a failure (of any kind), with the technique known as "implicit rejection". Secondly, the shared key K is now extracted not only from the "plaintext" $e = (e_0, e_1)$ but also from the ciphertext c (which is (c_0, c_1) in BIKE-1-CCA and BIKE-2-CCA). This allows to obtain a simpler security reduction in the CCA conversion, as we will mention later. Finally, BIKE-1-CCA and BIKE-3-CCA now require their "randomness" to be computed deterministically, as a hash of the plaintext: we use random oracles $\mathbf{G}$ and $\bar{\mathbf{G}}$ respectively.

2.3 Suggested Parameters

The parameters suggested in this section refer to the security levels indicated by NIST's call for papers, which relate to the hardness of a key search attack on a block cipher, like AES. More precisely, we indicate parameters for Levels 1, 3 and 5, corresponding to the security of AES-128, AES-192 and AES-256 respectively.

For the CPA secure variants, the parameters are chosen so that the One-Round Decoder described in Section 2.4.2 has a failure rate not exceeding 10^{-7} (validated through exhaustive simulation). Table 5 summarizes these three parameter suggestions. For the CCA secure variants, the parameters are chosen so that the Backflip Decoder described in Section 2.4.3 has the negligible failure rate required by the IND-CCA security proof (see Section 2.4.5 and Section 6.2).

	BIKE-1 and BIKE-2				BIKE-3				DFR[1]
Security	n	r	w	t	n	r	w	t	–
Level 1	20,326	10,163	142	134	22,054	11,027	134	154	10^{-7}
Level 3	39,706	19,853	206	199	43,366	21,683	198	226	10^{-7}
Level 5	65,498	32,749	274	264	72,262	36,131	266	300	10^{-7}

Table 4: Suggested Parameters for CPA Secure Variants.

	BIKE-1-CCA and BIKE-2-CCA				BIKE-3-CCA				DFR[2]
Security	n	r	w	t	n	r	w	t	–
Level 1	23,558	11,779	142	134	24,538	12,269	134	154	2^{-128}
Level 3	49,642	24,821	206	199	54,086	27,043	198	226	2^{-192}
Level 5	81,194	40,597	274	264	89,734	44,867	266	300	2^{-256}

Table 5: Suggested Parameters for CCA Secure Variants.

[1] DFR estimates for IND-CPA variants consider the One-Round Decoder (Section 2.4.2).
[2] DFR estimates for IND-CCA variants consider the Backflip Decoder (Section 2.4.3).

2.4 Decoding

2.4.1 Preliminaries

"Universal Interoperability": The decoder has a particular place in the specification. Its purpose is to find the unique solution of a decoding problem. Only one of the two parties needs to solve that problem and the way it does it will not change the view of the protocol –KEM and PKE alike.– The only exception to that is the decoding failure rate (DFR). But either one targets static keys and CCA security, the DFR has to be negligible, and failures never occur. Or one targets ephemeral keys and CPA security and a decoding failure will just abort the protocol, and the data which provoked the failure will be discarded. Thus, except for the DFR constraint for CCA security, the party in charge of decoding may select any decoder according to the best trade-off on its platform between algorithm cost (time/memory), easiness of implementation (software/hardware), side-channel resistance, ... This choice does not need to be known from the other party.

CPA *versus* CCA: We propose below two variants of the bit flipping algorithm corresponding to different trade-offs. The first one, "One Round Bit Flipping" was proposed for the first round BIKE proposal. It is currently the fastest and is suitable for the ephemeral key IND-CPA variant of BIKE. The second, "Backflipping", was developed to decrease the DFR estimates according to [40]. For a given security level, it features the smallest block size and is suitable for static key IND-CCA variants of BIKE.

Blackbox Decoder: In all variants of BIKE, we will consider the decoding as a black box running in bounded time and which either returns a valid error pattern or fails. It takes as arguments a (sparse) parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, a syndrome $s \in \mathbb{F}_2^{n-k}$, and an integer $u \geq 0$. Any returned value e is such that the Hamming distance between eH^T and s is smaller than u.

For given BIKE parameters r, w, t and variant, the key features are going to be the decoding time and the DFR (Decoding Failure Rate). Let $\mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$, the decoder input (H, s, u) is such that:

- the matrix H is block-circulant of index 2, that is a $H = (h_0^T \;\; h_1^T) \in \mathcal{R}^{1\times 2}$ such that $|h_0| = |h_1| = w/2$
- the integer u is either 0 (noiseless syndrome decoding, BIKE-1 and BIKE-2) or $t/2$ (noisy syndrome decoding, BIKE-3).

- the syndrome s is equal to $e' + e_0 h_0 + e_1 h_1$ for some triple $(e', e_0, e_1) \in \mathcal{R}^3$ such that $|e'| = u$ and $|e_0| + |e_1| = t$.

For each parameter set and each BIKE variant, the decoder input is entirely defined by h_0, h_1, e', e_0, e_1. The DFR is defined as the probability for the decoder to fail when the input (h_0, h_1, e', e_0, e_1) is distributed uniformly such that $|h_0| = |h_1| = w/2$, $|e'| = u$, and $|e_0| + |e_1| = t$.

2.4.2 One-Round Decoding

We will use the decoder defined in Algorithm 3. As it is defined, this algorithm returns a valid error pattern when it stops but it may not stop. In practice, A maximum running time is set, when this maximum is reached the algorithm stops with a failure. For given BIKE parameters r, w, and t, we have $n = 2r$ and $k = r$. In addition, we must (1) set values for S and δ and (2) provide a rule for computing the threshold (instruction 1).

Threshold Selection Rule. This rule derives from [10]. We use the notation of the algorithm, $s = eH^T$ is the input syndrome and e the corresponding (unknown) error. We denote $d = w/2$ the column weight of H. Let

$$\pi_1 = \frac{|s| + X}{td} \text{ and } \pi_0 = \frac{w\,|s| - X}{(n-t)d} \text{ where } X = \sum_{\ell \text{ odd}} (\ell - 1) \frac{r\binom{w}{\ell}\binom{n-w}{t-\ell}}{\binom{n}{t}}.$$

The counter value $|h_j \cap d|$ follows a distribution very close to a binomial distribution[3] $B(d, \pi_1)$ and $B(d, \pi_0)$ respectively if $e_j = 1$ or $e_j = 0$. From that it follows that the best threshold is the smallest integer T such that

$$t\binom{d}{T}\pi_1^T(1-\pi_1)^{d-T} \geq (n-t)\binom{d}{T}\pi_0^T(1-\pi_0)^{d-T},$$

that is (note that $\pi_1 \geq \pi_0$)

$$T = \left\lceil \frac{\log\frac{n-t}{t} + d\log\frac{1-\pi_0}{1-\pi_1}}{\log\frac{\pi_1}{\pi_0} + \log\frac{1-\pi_0}{1-\pi_1}} \right\rceil. \tag{1}$$

This value depends only of $n = 2r$, $w = 2d$, $t = |e|$ the error weight, and $|s|$ the syndrome weight. Details can be found in [10]. For any set of parameters thresholds can be precomputed.

In practice for a given set of parameters the formula (1) is very accurately approximated, in the relevant range for the syndrome weight, by an affine function:

[3] $B(n,p)$ the number of success out of n Bernoulli trials of probability p

Algorithm 3 `One-Round Bit Flipping Algorithm`

```
Require: $H \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$, integer $u \geq 0$
Ensure: $\left|s - eH^T\right| \leq u$
$T \leftarrow$ threshold($|s|$)
for $j = 0, \ldots, n-1$ do
    $\ell \leftarrow$ min(ctr($H, s, j$), $T$)
    $J_\ell \leftarrow J_\ell \cup \{j\}$                        // all $J_\ell$ empty initially
$e \leftarrow J_T$                                                (**)
$s' \leftarrow s - eH^T$
while $|s'| > S$ do                                               (***)
    for $\ell = 0, \ldots, \delta$ do                             (***)
        $e' \leftarrow$ check($H, s', J_{T-\ell}, d/2$)
        $(e, s') \leftarrow (e + e', s' - e'H^T)$                 // update error and syndrome
$e' \leftarrow$ check($H, s', e, d/2$)                            (**)
$(e, s') \leftarrow (e + e', s' - e'H^T)$                         // update error and syndrome
while $|s'| > u$ do
    $j \leftarrow$ guess_error_pos($H, s', d/2$)
    $(e_j, s') \leftarrow (e_j + 1, s' + h_j)$                    (*)
return $e$
```

```
check($H, s, J, T$)
    $e \leftarrow 0$
    for $j \in J$ do
        if ctr($H, s, j$) $\geq T$ then
            $e_j \leftarrow 1$
    return $e$
```

```
guess_error_pos($H, s, T$)
    loop                                  // until success
        $i \xleftarrow{\$} s$                          (**)
        for $j \in eq_i$ do                (*),(**)
            if ctr($H, s, j$) $\geq T$ then
                return $j$
```

```
ctr($H, s, j$)
    return $|h_j \cap s|$                  (*),(**)
```

```
threshold($S$)
    return function of $r$, $w$, $t$, and $S$
```

(*) *h_j the j-th column of H (as a row vector), eq_i the i-th row of H*

(**) *we identify binary vectors with the set of their non zero positions*

(***) *the algorithm uses two parameters S and δ which depend of r, w, and t*

- for BIKE-1 and BIKE-2
  - security level 1: $T = \lceil 13.530 + 0.0069722\ |s| \rceil$,
  - security level 3: $T = \lceil 15.932 + 0.0052936\ |s| \rceil$,
  - security level 5: $T = \lceil 17.489 + 0.0043536\ |s| \rceil$,
- for BIKE-3
  - security level 1: $T = \lceil 13.209 + 0.0060515\ |s| \rceil$,
  - security level 3: $T = \lceil 15.561 + 0.0046692\ |s| \rceil$,
  - security level 5: $T = \lceil 17.061 + 0.0038459\ |s| \rceil$.

2.4.3 Backflip Decoding

Algorithm 4 is akin to Algorithm 1 but to each flip it makes, it associates a time-to-live (a number of iterations during which it is kept). Flips reaching their end-of-life are reflipped at the beginning of each iteration. The thresholds are chosen similarly to previous algorithm assuming that all the previous flips removed an error. As in the preceding algorithm, it needs setting a maximum number of iterations.

Algorithm 4 `Backflipping Algorithm`

```
Require: $H \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$, integer $u \geq 0$
Ensure: $\left|s - eH^T\right| \leq u$
$e \leftarrow 0$ ; time $\leftarrow 1$ ; $F \leftarrow 0$                // $F_j$ = time-of-death of $j$
while $\left|s - eH^T\right| \leq u$ do
    for $j$ such that $F_j =$ time do      // Undo flips reaching their time-of-death
        $e_j \leftarrow 1 - e_j$ ; $F_j \leftarrow 0$
    $s' \leftarrow s - eH^T$
    $T \leftarrow$ threshold($|s'|, t - |F|$)
    time $\leftarrow$ time $+ 1$
    for $j \in \{0, \ldots, n-1\}$ do
        if $|s' \cap h_j| \geq T$ then                  // $h_j$ the $j$-th column of $H$
            $e_j \leftarrow 1 - e_j$ ; $F_j \leftarrow 0$ if $F_j \geq$ time else time $+$ ttl($|s' \cap h_j| - T$)
return $e$
```

Threshold Selection Rule $\texttt{threshold}(S, t')$**.** As before we define the following probabilities:

$$\pi_1 = \frac{S+X}{t'd} \text{ and } \pi_0 = \frac{wS-X}{(n-t')d} \text{ where } X = \sum_{\ell \text{ odd}} (\ell - 1) \frac{r\binom{w}{\ell}\binom{n-w}{t'-\ell}}{\binom{n}{t'}}.$$

Then $\texttt{threshold}(S, t')$ is the smallest T such that

$$t' \binom{d}{T} \pi_1^T (1-\pi_1)^{d-T} \geq (n-t') \binom{d}{T} \pi_0^T (1-\pi_0)^{d-T}.$$

Since t' is guessed optimistically (we assume that only errors are flipped in the algorithm: $t' = t - |F|$), it can happen that $\pi_1 > 1$. In this case $\texttt{threshold}(S, t')$ is the smallest T such that

$$1 \geq (n-t') \binom{d}{T} \pi_0^T (1-\pi_0)^{d-T}.$$

Time-to-live Rule $\texttt{ttl}(\delta)$**.** The time-to-live of a flip is determined using a saturating affine function in the difference between the counter value and the threshold,

- for BIKE-1 and BIKE-2
  - security level 1: $\texttt{ttl}(\delta) = \max(1, \min(5, \lfloor 1.1 + 0.45\,\delta \rfloor))$,
  - security level 3: $\texttt{ttl}(\delta) = \max(1, \min(5, \lfloor 1.41 + 0.36\,\delta \rfloor))$,
  - security level 5: $\texttt{ttl}(\delta) = \max(1, \min(5, \lfloor 1 + 0.45\,\delta \rfloor))$,
- for BIKE-3
  - security level 1: $\texttt{ttl}(\delta) = \max(1, \min(5, \lfloor 1.16 + 0.46\,\delta \rfloor))$,
  - security level 3: $\texttt{ttl}(\delta) = \max(1, \min(5, \lfloor 1.4 + 0.4\,\delta \rfloor))$,
  - security level 5: $\texttt{ttl}(\delta) = \max(1, \min(5, \lfloor 0.9 + 0.44\,\delta \rfloor))$.

2.4.4 Other Decoders

Other trade-offs are indeed possible, for instance soft decision decoders. They have more complex logic and arithmetic, but techniques as the Scaled Sum-Product Algorithm defined in [29] may lead to a lower DFR.

2.4.5 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, as stated in §5.1 the best known attacks are of order 2^{ct} or 2^{cw} with a constant c which only depends of the code rate (which can be either 1/2 or 1/3). This is corrected by a factor polynomial in r which is very small in practice. Moreover, if the block size varies, the code length and dimension vary in the same proportion and the code rate, thus the constant c in the exponent, remains the same. 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.

Estimating the DFR by Extrapolation. Therefore, increasing the block size while leaving the row weight w and the error weight t fixed is a valid strategy to reach lower DFR is needed.

A DFR as low as 2^{-256} is desirable for some levels of security. However measuring such a low probability is clearly out of reach from mere simulations.

In [40] the simple bit flipping Algorithm 5 was considered. In each one of its iterations, only one simple operation is performed. Using results from [10], a markovian model of this decoder was derived. This model allows for a fast estimation of the DFR of this algorithm for any set of parameters. In order to validate the model, the DFR obtained with an implementation of the algorithm and the values derived from the model were compared. It was observed that, while slightly optimistic, the model follows the same evolution as the simulations.

Algorithm 5 Step-by-Step Bit Flipping Algorithm

```
Require: $H \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$, integer $u \geq 0$
Ensure: $\left|s - eH^T\right| \leq u$
  $e \leftarrow 0$
  while $\left|s - eH^T\right| \leq u$ do
      $s' \leftarrow s - eH^T$
      $T \leftarrow \texttt{threshold}(context)$
      $j \leftarrow \texttt{sample}(context)$
      if $|s' \cap h_j| \geq T$ then
          $e_j \leftarrow 1 - e_j$
  return $e$
```

Increasing the range of values for the block size to include values were any failure is hardly observable with simulations, the following observations could be made on the logarithm of the DFR in function of the block size:

- it is always convex and strictly decreasing;
- it is superlinear in a first phase;
- it is then linear for higher block sizes.

Assuming the first property holds for any other “bit flipping-like” decoder, extrapolating the DFR can be done with a simple linear regression from the DFR for two different block sizes. Given the second property, the block sizes used for those two measures should be as high as possible. This is consistent with the asymptotic analysis of Theorem 1 and compatible with all simulations conducted so far.

2.5 Auxiliary Functions

Possible realizations of the auxiliary functions required by BIKE are described next. Other techniques can be used as long as they meet the target security level.

2.5.1 Pseudorandom Random Generators

Three types of pseudorandom bits stream generation are considered: no constraints on the output weight (Alg. 6), odd weight (Alg. 7), and specific weight w (Alg. 8). The common building block for them is AES-CTR-PRF based on AES-256, in CTR mode (following NIST SP800-90A guidelines [3]). For typical BIKE parameters the number of calls to AES with a given key is way below the restrictions on using AES in CTR mode. We remark that such AES-CTR-PRF generator is very efficient on modern processors equipped with dedicated AES instructions (e.g., AES-NI).

Algorithm 6 GenPseudoRand(seed, len)

```
Require: $\mathsf{seed}$ (32 bytes)
Ensure: $\bar{\mathsf{z}}$ ($\mathsf{len}$ pseudo-random bits $z$ embedded in array of bytes).
  $\mathsf{s} = \text{AES-CTR-INIT}(\mathsf{seed}, 0, 2^{32} - 1)$
  $\mathrm{z} = \mathsf{truncate}_{\mathsf{len}}\,(\text{AES-CTR-PRF}\,(\mathsf{s}, \mathsf{len}))$
  return $\bar{\mathsf{z}}$
```

2.5.2 Efficient Hashing

In this section, we describe a parallelized hash technique (see [18, 19, 21]) that can be used to accelerate the hashing process. We stress that a simple hash (e.g., SHA2 or SHA3 hash family) call can be used instead if (for interoperability reasons, for

Algorithm 7 GenPseudoRandOddWeight(seed, len)

```
Require: seed (32 bytes), len
Ensure: $\bar{z}$ (len pseudorandom bits $z$ of odd weight, in a byte array).
z = GenPseudoRand(seed, len)
if $weight(z)$ is even then z[0] = z[0] $\oplus 1$
return $\bar{z}$
```

Algorithm 8 WAES-CTR-PRF(s, wt, len)

```
Require: s (AES-CTR-PRF state), wt (32 bits), len
Ensure: A list (wlist) of wt bit-positions in $[0, \ldots, \text{len} - 1]$, updated s.
wlist= $\phi$; valid_ctr = 0
while valid_ctr < wt do
    $(\text{pos}, \text{s})$ = AES-CTR-PRF(s, 4)
    if ((pos < len) AND (pos $\notin$ wlist)) then
        wlist = wlist $\cup$ {pos}; valid_ctr = valid_ctr + 1
return wlist, s
```

instance) a standard hash function is preferred. Let hash be a hash function with digest length of ld bytes that uses a compression function compress which consumes a block of hbs bytes. The ParallelizedHash, with s slices, and pre-padding length srem, is described in Alg. 9. In our accompanying implementations, we instantiated hash with SHA-384.

3 Performance Analysis (2.B.2)

In this section, we discuss the performance of BIKE with respect to memory, latency and communication bandwidth. The performance numbers presented in sections 3.1, 3.2 and 3.3 refer to our reference code implementation, while section 3.7 refers to optimizations and their corresponding latency gains.

The platform used in the experiments was equipped with an Intel® Core™ i5-6260U CPU running at 1.80GHz. This platform has 32 GB RAM, 32K L1d and L1i cache, 256K L2 cache, and 4,096K L3 cache. Intel® Turbo Boost and Intel® Hyper-Threading technologies were all disabled. For each benchmark, the process was executed 25 times to warm-up the caches, followed by 100 iterations that were clocked (using the RDTSC instruction) and averaged. To minimize the effect of background tasks running on the system, each such experiment was repeated 10 times, and averaged. Our code was compiled using gcc/g++ 5.4.0 (build 20160609) with OpenSSL library (v1.0.2g, 1 Mar 2016) and NTL library (v6.2.1-1).

Algorithm 9 ParallelizedHash

```
Require: an array of la bytes $\mathsf{array}[\mathsf{la}-1:0]$, such that $\mathsf{la} \geq \mathsf{s} > 0$
Ensure: digest (ld bytes)
procedure ComputeSliceLen(la)
    tmp := $\mathsf{floor}\left(\frac{\mathsf{la}}{\mathsf{s}}\right) - \mathsf{slicerem}$
    $\alpha := \mathsf{floor}\left(\frac{\text{tmp}}{\mathsf{hbs}}\right)$
    return $\alpha \times \mathsf{hbs} + \mathsf{slicerem}$
procedure ParallelizedHash(array, $la$)
    ls := ComputeSliceLen(la)
    lrem := la - (ls × s)
    for $i := 0$ to (s -1) do
        $\mathsf{slice}[i] = \mathsf{array}[(i+1) \times \mathsf{ls} - 1 : i \times \mathsf{ls}]$
        $X[i] = \mathsf{hash}(\mathsf{slice}[i])$
    $Y = \mathsf{array}[\mathsf{la} - 1\text{: } \mathsf{ls} \times \mathsf{s}]$
    YX= Y $\|$ X[s − 1] $\|$ X[s − 2] ... $\|$ X[0]
    return hash(YX)
```

Regarding memory requirements, we remark that the IND-CPA BIKE private keys are composed by $(h_0, h_1) \in \mathcal{R}$ with $|h_0| = |h_1| = w/2$. Each element can either be represented by (r) bits or, in a more compact way, by the $w/2$ non-zero positions, yielding a $(\frac{w}{2} \cdot \lceil \log_2(r) \rceil)$-bits representation, thus the total private key size is $(w \cdot \lceil \log_2(r) \rceil)$-bits. Since $\lceil \log_2(r) \rceil < 16$ is true for all the proposed parameter sets, implementers may prefer (for the sake of simplicity) to store the private key as a sequence of w elements of 16-bits each. For the IND-CCA BIKE varaints, the private key is $(n + w \cdot \lceil \log_2(r) \rceil)$ bits long. The additional n bits are needed to store the (σ_0, σ_1) components. Depending on the application, users may want to explore the possibility of generating the private key on the fly from a cryptographically secure seed (memory vs. latency tradeoff).

3.1 Performance of BIKE-1

3.1.1 Memory Cost

Table 6 summarizes the memory required for each quantity.

3.1.2 Communication Bandwidth

Table 7 shows the bandwidth cost per message.

Quantity	Size	Level 1	Level 3	Level 5
Private key	$w \cdot \lceil \log_2(r) \rceil$	$1,988$	$3,090$	$4,110$
Public key	n	$20,326$	$39,706$	$65,498$
Ciphertext	n	$20,326$	$39,706$	$65,498$

Table 6: Private Key, Public Key and Ciphertext Size in Bits.

Message Flow	Message	Size	Level 1	Level 3	Level 5
Init. → Resp.	(f_0, f_1)	n	$20,326$	$39,706$	$65,498$
Resp. → Init.	(c_0, c_1)	n	$20,326$	$39,706$	$65,498$

Table 7: Communication Bandwidth in Bits.

3.1.3 Software Latency (Reference Implementation)

Operation	Level 1	Level 3	Level 5
Key Generation	$730,025$	$1,709,921$	$2,986,647$
Encapsulation	$689,193$	$1,850,425$	$3,023,816$
Decapsulation	$2,901,203$	$7,666,855$	$17,483,906$

Table 8: Latency Performance in Number of Cycles.

3.2 Performance of BIKE-2

3.2.1 Memory Cost

Table 9 summarizes the memory required for each quantity.

3.2.2 Communication Bandwidth

Table 10 shows the bandwidth cost per message.

Quantity	Size	Level 1	Level 3	Level 5
Private key	$w \cdot \lceil \log_2(r) \rceil$	$1,988$	$3,090$	$4,110$
Public key	r	$10,163$	$19,853$	$32,749$
Ciphertext	r	$10,163$	$19,853$	$32,749$

Table 9: Private Key, Public Key and Ciphertext Size in Bits.

Message Flow	Message	Size	Level 1	Level 3	Level 5
Init. → Resp.	f_1	r	$10,163$	$19,853$	$32,749$
Resp. → Init.	c	r	$10,163$	$19,853$	$32,749$

Table 10: Communication Bandwidth in Bits.

3.2.3 Software Latency (Reference Implementation)

Operation	Level 1	Level 3	Level 5
Key Generation	$6,383,408$	$22,205,901$	$58,806,046$
Encapsulation	$281,755$	$710,970$	$1,201,161$
Decapsulation	$2,674,115$	$7,114,241$	$16,385,956$

Table 11: Latency Performance in Number of Cycles.

3.3 Performance of BIKE-3

3.3.1 Memory Cost

Table 12 summarizes the memory required for each quantity.

3.3.2 Communication Bandwidth

Table 13 shows the bandwidth cost per message.

Quantity	Size	Level 1	Level 3	Level 5
Private key	$w \cdot \lceil \log_2(r) \rceil$	$1,876$	$2,970$	$4,256$
Public key	n	$22,054$	$43,366$	$72,262$
Ciphertext	n	$22,054$	$43,366$	$72,262$

Table 12: Private Key, Public Key and Ciphertext Size in Bits.

Message Flow	Message	Size	Level 1	Level 3	Level 5
Init. → Resp.	(f_0, f_1)	n	22,054	43,366	72,262
Resp. → Init.	(c_0, c_1)	n	22,054	43,366	72,262

Table 13: Communication Bandwidth in Bits.

3.3.3 Software Latency (Reference Implementation)

Operation	Level 1	Level 3	Level 5
Key Generation	$433,258$	$1,100,372$	$2,300,332$
Encapsulation	$575,237$	$1,460,866$	$3,257,675$
Decapsulation	$3,437,956$	$7,732,167$	$18,047,493$

Table 14: Latency Performance in Number of Cycles.

3.4 Performance of BIKE-1-CCA

3.4.1 Memory Cost

Table 15 summarizes the memory required for each quantity.

3.4.2 Communication Bandwidth

Table 16 shows the bandwidth cost per message.

Quantity	Size	Level 1	Level 3	Level 5
Private key	$n + w \cdot \lceil \log_2(r) \rceil$	$25,546$	$52,732$	$85,578$
Public key	n	$23,558$	$49,642$	$81,194$
Ciphertext	n	$23,558$	$49,642$	$81,194$

Table 15: Private Key, Public Key and Ciphertext Size in Bits.

Message Flow	Message	Size	Level 1	Level 3	Level 5
Init. → Resp.	(f_0, f_1)	n	$23,558$	$49,642$	$81,194$
Resp. → Init.	(c_0, c_1)	n	$23,558$	$49,642$	$81,194$

Table 16: Communication Bandwidth in Bits.

3.4.3 Software Latency (Reference Implementation)

Operation	Level 1	Level 3	Level 5
Key Generation	$973,689$	$2,716,851$	$6,076,775$
Encapsulation	$919,988$	$2,620,332$	$5,809,713$
Decapsulation	$6,617,747$	$16,252,967$	$35,174,191$

Table 17: Latency Performance in Number of Cycles.

3.5 Performance of BIKE-2-CCA

3.5.1 Memory Cost

Table 18 summarizes the memory required for each quantity.

3.5.2 Communication Bandwidth

Table 19 shows the bandwidth cost per message.

Quantity	Size	Level 1	Level 3	Level 5
Private key	$n + w \cdot \lceil \log_2(r) \rceil$	$25,546$	$52,732$	$85,578$
Public key	r	$11,779$	$24,821$	$40,597$
Ciphertext	r	$11,779$	$24,821$	$40,597$

Table 18: Private Key, Public Key and Ciphertext Size in Bits.

Message Flow	Message	Size	Level 1	Level 3	Level 5
Init. → Resp.	f_1	r	$11,779$	$24,821$	$40,597$
Resp. → Init.	c	r	$11,779$	$24,821$	$40,597$

Table 19: Communication Bandwidth in Bits.

3.5.3 Software Latency (Reference Implementation)

Operation	Level 1	Level 3	Level 5
Key Generation	$8,091,390$	$32,059,772$	$67,737,696$
Encapsulation	$478,921$	$1,314,762$	$2,977,652$
Decapsulation	$5,821,836$	$13,840,081$	$29,466,909$

Table 20: Latency Performance in Number of Cycles.

3.6 Performance of BIKE-3-CCA

3.6.1 Memory Cost

Table 21 summarizes the memory required for each quantity.

3.6.2 Communication Bandwidth

Table 22 shows the bandwidth cost per message.

Quantity	Size	Level 1	Level 3	Level 5
Private key	$n + w \cdot \lceil \log_2(r) \rceil$	26,414	57,056	93,990
Public key	n	24,538	54,086	89,734
Ciphertext	n	24,538	54,086	89,734

Table 21: Private Key, Public Key and Ciphertext Size in Bits.

Message Flow	Message	Size	Level 1	Level 3	Level 5
Init. → Resp.	(f_0, f_1)	n	24,538	54,086	89,734
Resp. → Init.	(c_0, c_1)	n	24,538	54,086	89,734

Table 22: Communication Bandwidth in Bits.

3.6.3 Software Latency (Reference Implementation)

Operation	Level 1	Level 3	Level 5
Key Generation	600,088	1,736,834	3,856,747
Encapsulation	935,794	2,885,347	6,728,690
Decapsulation	7,043,845	16,931,150	36,502,736

Table 23: Latency Performance in Number of Cycles.

3.7 Optimizations and Performance Gains

Optional algorithmic optimizations for BIKE and the corresponding performance gains are discussed next.

3.7.1 BIKE-2 Batch Key Generation (Optimized Implementation)

BIKE-2 key generation needs to compute a (costly) polynomial inversion, as described in Section 2.1.2. To reduce the impact of this costly operation and still benefit from the lower communication bandwidth offered by BIKE-2, we propose a *batch* version of BIKE-2 key generation. The main benefit of this approach is that only one polynomial inversion is computed for every N key generations, assuming a predefined $N \in \mathbb{N}$, instead of one inversion per key generation.

This technique is based on Montgomery's trick [37] and assumes that multiplication is fairly less expensive than inversion. As a toy example, suppose that one needs to invert two polynomials $x, y \in \mathcal{R}$. Instead of computing the inverse of each one separately, it is possible to compute them with one inversion and three multiplications: set $tmp = x \cdot y$, $inv = tmp^{-1}$ and then recover $x^{-1} = y \cdot inv$ and $y^{-1} = x \cdot inv$. This can be easily generalized to $N > 2$ polynomials: in this case, $2N$ multiplications are needed and inverses need to be recovered one at a time and in order. Because of this, our implementation requires the maintenance of a global variable $0 \leq \texttt{keyindex} < N$ that must be accessible only to the legitimate party willing to generate BIKE-2 keys and increased after each key generation. Algorithm 10 describes this optimization. Most of the work is done in the first key generation ($\texttt{keyindex} = \texttt{0}$). In this way, the amortized cost of BIKE-2 key generation is reduced significantly as illustrated in Table 24 and Table 25.

Algorithm 10 `BIKE-2 Batch Key Generation`

```
Require: keyindex, $N \in \mathbb{N}$, code parameters $(n, k, w)$
Ensure: $(h_{0,0}, \ldots, h_{0,N-1}, h_1) \in \mathcal{R}^{N+1}$, $|h_{0,i}|_{0 \leq i < N} = |h_1| = w$
Sample $h_1 \xleftarrow{\$} \mathcal{R}$ such that $|h_1| = w$
if keyindex = 0 then
    Sample $h_{0,i} \xleftarrow{\$} \mathcal{R}$ such that $|h_{0,i}| = w$ for $0 < i < N$
    $\texttt{prod}_{\texttt{0,0}} = \texttt{h}_{\texttt{0,0}}$
    $\texttt{prod}_{\texttt{0,i}} = \texttt{prod}_{\texttt{0,i-1}} \cdot \texttt{h}_{\texttt{0,i}}$, for $1 \leq i < N$
    $\texttt{prod}_{\texttt{1,N-1}} = \texttt{prod}_{\texttt{0,N-1}}^{-1}$
    $\texttt{prod}_{\texttt{1,i}} = \texttt{prod}_{\texttt{1,i+1}} \cdot \texttt{h}_{\texttt{0,i+1}}$, for $N - 2 \geq i > 0$
    $inv = \texttt{prod}_{\texttt{1,1}} \cdot \texttt{h}_{\texttt{0,1}}$
else
    $inv = \texttt{prod}_{\texttt{1,keyindex}} \cdot \texttt{prod}_{\texttt{0,keyindex-1}}$
$h \leftarrow h_1 \cdot inv$
keyindex ← keyindex + 1
return $(h_{0,\texttt{keyindex}}, h_1, h)$
```

Operation	Reference	Batch	Gain (%)
Level 1	$6,383,408$	$1,647,843$	74.18%
Level 3	$22,205,901$	$4,590,452$	79.32%
Level 5	$58,806,046$	$9,296,144$	84.19%

Table 24: BIKE-2 Batch Key Generation Performance Gain (in cycles, for $N = 100$).

Operation	Reference	Batch	Gain (%)
Level 1	$8,091,390$	$1,756,545$	78.29%
Level 3	$32,059,772$	$5,171,501$	83.86%
Level 5	$67,737,696$	$11,795,658$	82.58%

Table 25: BIKE-2-CCA Batch Key Generation Performance Gain (in cycles, for $N = 100$).

We stress that an implementer interested in the benefits offered by BIKE-2 batch key generation will need to meet the additional security requirements of protecting from adversaries and securely updating the variables `keyindex`, $prod_0$ and $prod_1$. It is also important to stress that the keys generated through this batch process are not related to each other. Finally, we remark that the use (or not) of the batch optimization does not impact on the encapsulation and decapsulation processes described in Section 2.1.2.

3.8 Additional Software Implementation

To illustrate the potential performance that BIKE code may achieve when running on modern platforms, we report some results of an additional implementation. These preliminary BIKE-1 and BIKE-2 results can be expected to be further improved.

The performance is reported in processor cycles (lower is better), reflecting the performance per a *single core*. The results were obtained with the same measurement methodology declared in Section 3. The results are reported in Tables 26, 27, and 28 for BIKE-1, and in Tables 29, 30, and 31 for BIKE-2.

The additional implementation code. The core functionality was written in x86 assembly, and wrapped by assisting C code. The implementations use the PCLMULQDQ, AES−NI and the AVX2 and AVX512 architecture extensions. The code was compiled with gcc (version 5.4.0) in 64-bit mode, using the "O3" Optimization level, and run on a Linux (Ubuntu 16.04.3 LTS) OS. Details on the implementation and optimized components are provided in [16], and the underlying primitives are available in [20].

The benchmarking platform. The experiments were carried out on a platform equipped with the latest 8^{th} Generation Intel® Core^{TM} processor ("Kaby Lake") - Intel® Xeon® Platinum 8124M CPU at 3.00 GHz Core® i5 – 750. The platform has 70 GB RAM, 32K L1d and L1i cache, 1,024K L2 cache, and 25,344K L3 cache. It was configured to disable the Intel® Turbo Boost Technology, and the Enhanced Intel Speedstep® Technology.

	—			Constant time implementation		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
AVX2	0.09	0.11	1.13	0.20	0.15	5.30
AVX512	0.09	0.11	1.02	0.19	0.13	4.86

Table 26: Performance (in millions of cycles) of BIKE-1 Level 1.

	—			Constant time implementation		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
AVX2	0.25	0.28	3.57	0.45	0.36	16.74
AVX512	0.25	0.27	2.99	0.45	0.33	15.26

Table 27: Performance (in millions of cycles) of BIKE-1 Level 3.

	—			Constant time implementation		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
AVX2	11.99	0.27	2.70	12.45	0.39	10.74
AVX512	11.99	0.25	2.14	12.34	0.34	8.93

Table 31: Performance (in millions of cycles) of BIKE-2 Level 5.

	—			Constant time implementation		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
AVX2	0.25	0.29	2.75	0.67	0.42	9.84
AVX512	0.25	0.27	2.24	0.69	0.36	8.27

Table 28: Performance (in millions of cycles) of BIKE-1 Level 5.

	—			Constant time implementation		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
AVX2	4.38	0.09	1.12	4.46	0.12	5.55
AVX512	4.38	0.08	0.86	4.45	0.11	5.12

Table 29: Performance (in millions of cycles) of BIKE-2 Level 1.

3.9 Hardware Implementation

For our hardware reference design we assume a common use-case of an embedded device communicating with an server or cloud infrastructure. In this setting a hardware device represents only one endpoint in the communcation so that we provide hardware design that features the required `KeyGen` and `Encaps` of BIKE. Specifically, we implemented `KeyGen` and `Encaps` of BIKE-1, Level 1 on a Xilinx Artix-7 (xc7a35tcpg236-1) FPGA following a similar concept as proposed in [42] but took advantage of additional optimizations enabled by the BIKE construction. All details are listed in the below.

3.9.1 Reference Implementation

The first implementation of `KeyGen` and `Encaps` is based on the approach from [42] which was optimized to achieve a small hardware footprint. The encoding step is performed by storing the key in one true dual-port BRAM. Each clock cycle 64 bit of the key are read from the memory, shifted by one bit and are wrote back to the memory. This will not only lead to a one bit shifted 32 bit word but also to a shift of the content by one entire address. However, when reading 64 bit of the key, these bits are added to the current intermediate results, which are also stored in a true dual-port BRAM. Before the encoding is started, the error is already stored in that memory such that a final addition of the error is not necessary.

	—			Constant time implementation		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
AVX2	7.77	0.17	2.88	8.04	0.27	17.36
AVX512	7.79	0.18	3.48	8.05	0.23	15.63

Table 30: Performance (in millions of cycles) of BIKE-2 Level 3.

The reference design of `KeyGen` follows a straightforward implementation using a non-constant time sampling operation. Since the generation of the public key also requires an encoding step, the same module as for `Encaps` is used. The memory layout for the keys and for the error is done by storing the LSB at address zero. This is reversed for the sampled message m and g. Here the MSB is stored at address zero.

Table 32: Reference Implementation of `KeyGen` plus `Encaps` (excluding hash).

	Resources			
	Logic	Memory	Area	BRAM
	LUT	FF	Slices	Tiles
BIKE-1, Level 1	918	236	276	5
KeyGen				3
Sample SK	134	38	58	1
Sample G	88	30	32	1
Compute PK	130	51	78	1
Encaps	366	103	125	2
Sample E	68	20	35	1
Sample M	23	16	10	1
Encoding	141	60	87	0

Table 32 shows the implementation results for the state-of-the-art implementation excluding the additional hash function. In summary, this implementation requires 276 Slices and 5 BRAMs of the available hardware resources. To finish `KeyGen` and `Encaps`, the implementation requires 6 504 975 clock cycles. The maximum usable frequency turned out to be 162.6 MHz which leads to a 40.01 ms delay from starting `KeyGen` to finishing `Encaps`.

In Table 33 we summarized the implementation results for the state-of-the-art implementation using SHA-384 as hash function. The footprint increases from

276 Slices to 1 242 Slices and it requires one more BRAM module to store the sampled error. Since the hash function can be freely chosen and is not optimized in our design, this implementation results should only be understand as an example.

Table 33: Reference Implementation of `KeyGen` plus `Encaps` (including hash).

	Resources			
	Logic	Memory	Area	BRAM
	LUT	FF	Slices	Tiles
BIKE-1, Level 1	4 449	2 765	1 242	6
KeyGen				3
Sample SK	134	38	60	1
Sample G	89	30	37	1
Compute PK	130	51	88	1
Encaps	3 896	2 632	1 118	3
Sample E	68	20	42	1
Sample M	23	16	10	1
Encoding	141	60	92	0
SHA-384	3 534	2 529	1 002	1

3.9.2 Improvement I

BIKE enables a scalable optimization a trade-off between resource cost and performance. To improve performance we split up the keys in a first optimization level and used one true dual-port BRAM to store each lower part of the keys and the error (f_0, h_0, e_0) and one true dual-port BRAM to store each upper part of the keys and the error (f_1, h_1, e_1). This allows us to work with an increased data-bus of 128 bit for each key and the sampled error. Since we only use one single bit of g and m for one row of the keys, we do not need to double up the memory for these polynomials.

The implementation results of this first level of optimization can be found in Table 34 and Table 35. The first table provides the results without using any hash function and the second one uses SHA-384 again. Excluding the hash function this results in a hardware footprint of 379 Slices and 8 BRAMs which increases the utilization by only 37.32 % while decreasing the required clock cycles by 50.32 % to 3 273 144. We determined the maximum frequency to 161.3 MHz which allows us to finish `KeyGen` and `Encaps` within 20.29 ms.

Table 34: Optimized Results (Level 1) for `KeyGen` plus `Encaps` (excluding hash).

	Resources			
	Logic	Memory	Area	BRAM
	LUT	FF	Slices	Tiles
BIKE-1, Level 1	1 233	231	379	8
KeyGen				5
Sample SK	134	38	62	2
Sample G	89	30	38	1
Compute PK	187	49	114	2
Encaps	518	100	190	3
Sample E	102	20	51	2
Sample M	23	16	10	1
Encoding	195	57	123	0

Table 35: Optimized Results (Level 1) for `KeyGen` plus `Encaps` (including hash).

	Resources			
	Logic	Memory	Area	BRAM
	LUT	FF	Slices	Tiles
BIKE-1, Level 1	4 814	2 760	1 381	9
KeyGen				5
Sample SK	134	38	73	2
Sample G	88	30	35	1
Compute PK	187	49	111	2
Encaps	4 096	2 629	1 180	4
Sample E	147	20	74	2
Sample M	23	16	11	1
Encoding	195	57	120	0
SHA-384	3 533	2 529	1 012	1

3.9.3 Improvement II

In the second level of optimization we doubled the allocated memory for the secret and public key. After the sampling of the secret key and after computing the public key, we performed after both operations a preparatory shift of the keys by

$$\frac{\lceil \texttt{R\_BITS}/32 \rceil}{2} \cdot 32 = 5\,088\,\text{bits}. \tag{2}$$

The shifted keys will be denoted by sk' and pk' respectively. Using a true dual-port BRAM for g and m as well, allows us to read out, multiply and shift the upper and lower part of the circulant matrices in parallel. Since the multiplication is now based on two bits of g or m, we have four different cases to calculate the new intermediate result I_{new}. The following example shows them for the multiplication of the secrete key with g which is controlled by the two bits g_i and g_j of g .

$$I_{new} = \begin{cases} I_{old} & \text{when } g_i = 0; g_j = 0 \\ I_{old} \oplus sk_{old} & \text{when } g_i = 1; g_j = 0 \\ I_{old} \oplus sk'_{old} & \text{when } g_i = 0; g_j = 1 \\ I_{old} \oplus sk_{old} \oplus sk'_{old} & \text{when } g_i = 1; g_j = 1 \end{cases} \tag{3}$$

This optimization technique increases the hardware utilization by 62.27 % in terms of slices and requires four additional BRAM tiles compared to the first optimized design. The number of required clock cycles decreases from 3 273 144 to 1 639 461. Using a maximum frequency of 151.5 MHz, finishes `KeyGen` and `Encaps` within 10.82 ms. This is roughly a speed up by a factor of four compared to the state-of-the-art implementation. As well as for the previous approaches, we provide an implementation including a SHA-384 core. The result is shown in Table 37.

Table 36: Optimized Results (Level 2) for `KeyGen` plus `Encaps` (excluding hash).

	Resources			
	Logic	Memory	Area	BRAM
	LUT	FF	Slices	Tiles
BIKE-1, Level 1	1 895	357	615	12
KeyGen				9
Sample SK	134	38	59	2
Sample G	88	29	38	1
Compute PK	349	50	181	2
Preparatory Shifts	154	134	82	4
Encaps	668	89	213	3
Sample E	95	16	39	2
Sample M	23	16	9	1
Encoding	343	50	155	0

Table 37: Optimized Results (Level 2) for `KeyGen` plus `Encaps` (including hash).

	Resources			
	Logic	Memory	Area	BRAM
	LUT	FF	Slices	Tiles
BIKE-1, Level 1	5 465	2 886	1 559	13
KeyGen				9
Sample SK	134	38	68	2
Sample G	88	29	34	1
Compute PK	348	50	160	2
Preparatory Shifts	154	134	91	4
Encaps	4 235	2 618	1 218	4
Sample E	140	16	72	2
Sample M	23	16	8	1
Encoding	343	50	171	0
SHA-384	3 522	2 529	994	1

3.9.4 Comparison

In Table 38 we provide a concluding overview about all six implementations. It is shown that BIKE including further optimization can outperform our reference (and previously reported implementations) by a factor of four in terms of clock cycles. This is remarkable since the improvement is reached by only spending 2.23 times more slices and 12 instead of 5 BRAM tiles, respectively.

Table 38: Comparison of all implementations.

	Resources		**Throughput**		
	Area	BRAM	Clock Cycles	Frequency	Time
	Slices	Tiles	CC	MHz	ms
Reference	276	5	6 504 975	162.6	40.01
Reference + Hash	1 242	6	6 504 975	163.9	39.68
Opt. Level 1	379	8	3 273 144	161.3	20.29
Opt. Level 1 + Hash	1 381	9	3 273 144	161.3	20.29
Opt. Level 2	615	12	1 639 461	151.5	10.82
Opt. Level 2 + Hash	1 559	13	1 639 461	161.3	10.16

3.9.5 Enhanced Levels of Optimization

The generic optimization technique discussed in Section 3.9.3 can be continuously applied for increased performance at higher resource cost. According to Equation 3 the required hardware resources is estimated to grow by a quadratic factor. A rough estimation about the required hardware resources depending on the number of clock cycles is shown in Figure 1.

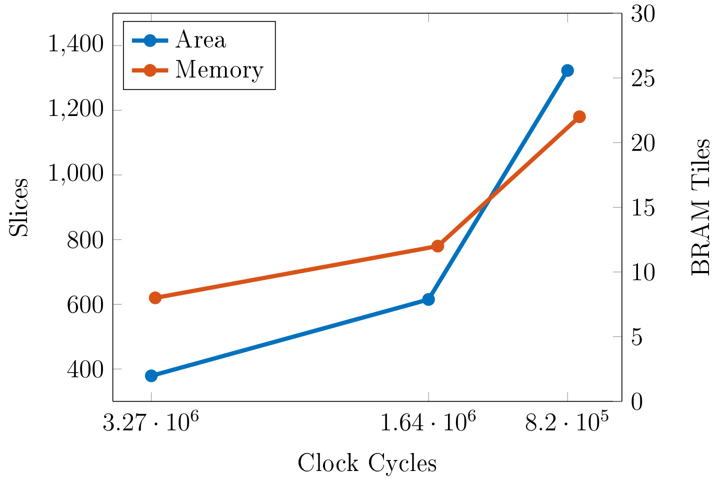


Figure 1: Estimation for further levels of optimization.

4 Known Answer Values – KAT (2.B.3)

4.1 IND-CPA Variants

4.1.1 KAT for BIKE-1

The KAT files of BIKE-1 are available in:

- req file: `KAT/INDCPA/BIKE1/PQCkemKAT_BIKE1-Level1_2542.req`
- rsp file: `KAT/INDCPA/BIKE1/PQCkemKAT_BIKE1-Level1_2542.rsp`
- req file: `KAT/INDCPA/BIKE1/PQCkemKAT_BIKE1-Level3_4964.req`
- rsp file: `KAT/INDCPA/BIKE1/PQCkemKAT_BIKE1-Level3_4964.rsp`
- req file: `KAT/INDCPA/BIKE1/PQCkemKAT_BIKE1-Level5_8188.req`
- rsp file: `KAT/INDCPA/BIKE1/PQCkemKAT_BIKE1-Level5_8188.rsp`

4.1.2 KAT for BIKE-2

The KAT files of BIKE-2 are available in:

- req file: `KAT/INDCPA/BIKE2/PQCkemKAT_BIKE2-Level1_2542.req`
- rsp file: `KAT/INDCPA/BIKE2/PQCkemKAT_BIKE2-Level1_2542.rsp`
- req file: `KAT/INDCPA/BIKE2/PQCkemKAT_BIKE2-Level3_4964.req`
- rsp file: `KAT/INDCPA/BIKE2/PQCkemKAT_BIKE2-Level3_4964.rsp`
- req file: `KAT/INDCPA/BIKE2/PQCkemKAT_BIKE2-Level5_8188.req`
- rsp file: `KAT/INDCPA/BIKE2/PQCkemKAT_BIKE2-Level5_8188.rsp`

4.1.3 KAT for BIKE-3

The KAT files of BIKE-3 are available in:

- req file: `KAT/INDCPA/BIKE3/PQCkemKAT_BIKE3-Level1_2758.req`
- rsp file: `KAT/INDCPA/BIKE3/PQCkemKAT_BIKE3-Level1_2758.rsp`
- req file: `KAT/INDCPA/BIKE3/PQCkemKAT_BIKE3-Level3_5422.req`
- rsp file: `KAT/INDCPA/BIKE3/PQCkemKAT_BIKE3-Level3_5422.rsp`
- req file: `KAT/INDCPA/BIKE3/PQCkemKAT_BIKE3-Level5_9034.req`
- rsp file: `KAT/INDCPA/BIKE3/PQCkemKAT_BIKE3-Level5_9034.rsp`

4.2 IND-CCA Variants

4.2.1 KAT for BIKE-1-CCA

The KAT files of BIKE-1-CCA are available in:

- req file: `KAT/INDCCA/BIKE1/PQCkemKAT_BIKE1-Level1_8838.req`
- rsp file: `KAT/INDCCA/BIKE1/PQCkemKAT_BIKE1-Level1_8838.rsp`
- req file: `KAT/INDCCA/BIKE1/PQCkemKAT_BIKE1-Level3_18618.req`
- rsp file: `KAT/INDCCA/BIKE1/PQCkemKAT_BIKE1-Level3_18618.rsp`
- req file: `KAT/INDCCA/BIKE1/PQCkemKAT_BIKE1-Level5_30450.req`
- rsp file: `KAT/INDCCA/BIKE1/PQCkemKAT_BIKE1-Level5_30450.rsp`

4.2.2 KAT for BIKE-2-CCA

The KAT files of BIKE-2-CCA are available in:

- req file: `KAT/INDCCA/BIKE2/PQCkemKAT_BIKE2-Level1_8838.req`
- rsp file: `KAT/INDCCA/BIKE2/PQCkemKAT_BIKE2-Level1_8838.rsp`
- req file: `KAT/INDCCA/BIKE2/PQCkemKAT_BIKE2-Level3_18618.req`
- rsp file: `KAT/INDCCA/BIKE2/PQCkemKAT_BIKE2-Level3_18618.rsp`
- req file: `KAT/INDCCA/BIKE2/PQCkemKAT_BIKE2-Level5_30450.req`
- rsp file: `KAT/INDCCA/BIKE2/PQCkemKAT_BIKE2-Level5_30450.rsp`

4.2.3 KAT for BIKE-3-CCA

The KAT files of BIKE-3-CCA are available in:

- req file: `KAT/INDCCA/BIKE3/PQCkemKAT_BIKE3-Level1_9204.req`
- rsp file: `KAT/INDCCA/BIKE3/PQCkemKAT_BIKE3-Level1_9204.rsp`
- req file: `KAT/INDCCA/BIKE3/PQCkemKAT_BIKE3-Level3_20286.req`
- rsp file: `KAT/INDCCA/BIKE3/PQCkemKAT_BIKE3-Level3_20286.rsp`
- req file: `KAT/INDCCA/BIKE3/PQCkemKAT_BIKE3-Level5_33654.req`
- rsp file: `KAT/INDCCA/BIKE3/PQCkemKAT_BIKE3-Level5_33654.rsp`

5 Known Attacks (2.B.5)

This section discusses the practical security aspects of our proposal.

5.1 Hard Problems and Security Reduction

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

Problem 1 (Syndrome Decoding – SD).
Instance: $H \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$, *an integer* $t > 0$.
Property: *There exists* $e \in \mathbb{F}_2^n$ *such that* $|e| \leq t$ *and* $eH^T = s$.

Problem 2 (Codeword Finding – CF)**.**
Instance: $H \in \mathbb{F}_2^{(n-k)\times n}$*, an integer* $t > 0$*.*
Property: *There exists* $c \in \mathbb{F}_2^n$ *such that* $|c| = t$ *and* $cH^T = 0$*.*

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

5.1.1 Hardness for QC codes.

Coding problems (SD and CF) in a QC-code are NP-complete, but the result does not hold for when the index is fixed. In particular, for $(2, 1)$-QC codes or $(3, 1)$-QC codes, which are of interest to us, we do not know whether or not SD and CF are NP-complete.

Nevertheless, they are believed to be hard on average (when r grows) and the best solvers in the quasi-cyclic case have the same cost as in the generic case up to a small factor which never exceeds the order r of quasi-cyclicity. The problems below are written in the QC setting, moreover we assume that the parity check matrix H is in systematic form, that is the first $(n_0 - k_0) \times (n_0 - k_0)$ block of H is the identity matrix. For instance, for $(2, 1)$-QC and $(3, 1)$-QC codes codes, the parity check matrix (over $\mathcal{R}$) respectively have the form

$$\begin{pmatrix} 1 & h \end{pmatrix} \text{ with } h \in \mathcal{R}, \text{ and } \begin{pmatrix} 1 & 0 & h_0 \\ 0 & 1 & h_1 \end{pmatrix} \text{ with } h_0, h_1 \in \mathcal{R}.$$

In our case, we are interested only by those two types of QC codes and to the three related hard problems below:

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

Problem 4 ($(2, 1)$-QC Codeword Finding – $(2, 1)$-QCCF)**.**
Instance: h *in* $\mathcal{R}$*, an integer* $t > 0$*.*
Property: *There exists* c_0, c_1 *in* $\mathcal{R}$ *such that* $|c_0| + |c_1| = t$ *and* $c_0 + c_1 h = 0$*.*

Problem 5 ($(3,1)$-QC Syndrome Decoding – $(3,1)$-QCSD)**.**
Instance: s_0, s_1, h_0, h_1 *in* $\mathcal{R}$*, an integer* $t > 0$*.*
Property: *There exists* e_0, e_1, e_2 *in* $\mathcal{R}$ *such that* $|e_0|+|e_1|+|e_2| \leq 3t/2$*,* $e_0+e_2h_0 = s_0$ *and* $e_1 + e_2h_1 = s_1$*.*

In the decisional variant of the (2,1)-QCSD problem, an adversary has to decide for appropriately sampled (s, h) whether there exists an error that matches the expected property. Due to the restriction on the weight of the sampling, this leads to sampling h uniformly with an odd weight, and s with an even weight. For the (3,1)-QCSD, we focus on the sampling of s_0, s_1, h_0, h_1, where h_0, h_1 are even, s_0 is random with the same parity of $t/2$, and s_1 is random and even. As they are presented, those problems have the appearance of *sparse polynomials* problem, but in fact they are equivalent to the generic quasi-cyclic decoding and codeword finding problems.

In the current state of the art, the best known techniques for solving those problems are variants of Prange's Information Set Decoding (ISD) [38]. We remark that, though the best attacks consist in solving one of the search problems, the security reduction of our scheme requires the decision version of Problem 2.

5.2 Information Set Decoding

The best asymptotic variant of ISD is due to May and Ozerov [34], but it has a polynomial overhead which is difficult to estimate precisely. In practice, the BJMM variant [5] is probably the best for relevant cryptographic parameters. The work factor for classical (*i.e.* non quantum) computing of any variant $\mathcal{A}$ of ISD for decoding t errors (or finding a word of weight t) in a binary code of length n and dimension k can be written

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

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 $w \approx t \approx \sqrt{n}$), we have $c = \log_2 \frac{1}{1-R}$ 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 [23] gives $\mathrm{WF}_{\mathrm{BJMM}}(65542, 32771, 264) = 2^{263.3}$ "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.

5.2.1 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) [39] 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.

5.2.2 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 5. *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.

This proposition is proved in Section B of the appendix. The point is that (essentially) the best quantum attack on our scheme consists in using Grover's search on the information sets computed in Prange's algorithm (this is Bernstein's algorithm [7]). Theoretically there is a slightly better algorithm consisting in quantizing more sophisticated ISD algorithms [27], 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.

5.3 Defeating the GJS Reaction Attack

BIKE IND-CPA variants use ephemeral KEM key pairs, i.e. a KEM key generation is performed for each key exchange. As a result, the GJS reaction attack is inherently defeated: a GJS adversary would have (at most) a single opportunity to observe decryption, thus not being able to create statistics about different error patterns. We note that, for efficiency purposes, an initiator may want to precompute KEM key pairs before engaging in key exchange sessions. We remark that policies to securely store the pregenerated KEM key pair must be in place, in order to avoid that an adversary access a KEM key pair that is going to be used in a future communication.

5.4 Choice of Parameters

We denote $\mathrm{WF}(n, k, t)$ the workfactor of the best ISD variant for decoding t errors in a binary code of length n and dimension k. In the following we will consider only codes of transmission rate 0.5, that is length $n = 2r$ and dimension r. In a classical setting, the best solver for Problem 3 has a cost $\mathrm{WF}(2r, r, t)/\sqrt{r}$, the best solver for Problem 4 has a cost $\mathrm{WF}(2r, r, w)/r$, and the best solver for Problem 5 has a cost $\mathrm{WF}(3r, r, 3t/2)/\sqrt{r}$. As remarked above, with $\mathrm{WF}(n, k, \ell) \approx 2^{\ell \log_2 \frac{n}{n-k}}$ we obtain a crude but surprisingly accurate, parameter selection rule. We target security levels corresponding to AES λ with $\lambda \in \{128, 192, 256\}$. To reach λ bits of classical security, we choose w, t and r such that

- for BIKE-1 and BIKE-2, Problem 3 with block size r and weight t and Problem 4 with block size r and weight w must be hard enough, that is

$$\lambda \approx t - \frac{1}{2}\log_2 r \approx w - \log_2 r. \tag{4}$$

- for BIKE-3, Problem 5 with block size r and weight $3t/2$ and Problem 3 with block size r and weight w must be hard enough, that is

$$\lambda \approx \frac{3t}{2}\log_2 \frac{3}{2} - \frac{1}{2}\log_2 r \approx w - \frac{1}{2}\log_2 r. \tag{5}$$

Those equation have to be solved in addition with the constraint that r must be large enough to decode t errors in $(2, 1, r, w)$-QC-MDPC code with a negligible failure rate. Finally, we choose r such that 2 is primitive modulo r. First, this will force r to be prime, thwarting the so-called squaring attack [30]. Also, it implies that $(X^r - 1)$ only has two irreducible factors (one of them being $X - 1$). This is an insurance against an adversary trying to exploit the structure of $\mathbb{F}_2[X]/\langle X^r - 1\rangle$ when $(X^r - 1)$ has small factors, other than $(X - 1)$. This produces the parameters proposed in the document.

The quantum speedup is at best quadratic for the best solvers of the problems on which our system, from the arguments of §5.2.2, it follows our set of parameters correspond the security levels 1, 3, and 5 described in the NIST call for quantum safe primitives.

6 Formal Security (2.B.4)

6.1 IND-CPA Security

In this section we show that BIKE-1/2/3 variants are IND-CPA secure.

We start with the following definition where we denote by $\mathcal{K}$ the domain of the exchanged symmetric keys and by λ the security level.

Definition 6. *A key-encapsulation mechanism is IND-CPA (passively) secure if, for any polynomial-time adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following game is negligible.*

```
**Game** IND-CPA
1: $(sk, pk) \leftarrow \textsc{Gen}(\lambda)$
2: $(c, K_0) \leftarrow \textsc{Encaps}(pk)$
3: $K_1 \xleftarrow{\$} \mathcal{K}$
4: $c^* \leftarrow c$
5: $K^* \leftarrow K_b$
6: $b' \leftarrow \mathcal{A}(pk, c^*, K^*)$
```

We define the adversary's advantage as $\mathrm{Adv}^{\textit{IND-CPA}}(\mathcal{A}) = Pr[b' = b] - 1/2$.

Theorem 2.

1. *BIKE-1,2 are IND-CPA secure in the Random Oracle Model under the* $(2,1)$*-QCCF and* $(2,1)$*-QCSD assumptions.*
2. *BIKE-3 is IND-CPA secure in the Random Oracle Model under the* $(3,1)$*-QCSD and* $(2,1)$*-QCSD assumptions.*

Preliminary Remarks on Parity and Balancedness: In the proof of Theorem 2, we will use decisional versions of the $(2,1)$-QCSD, $(2,1)$-QCCF, and $(3,1)$-QCSD problems, instead of their search versions given in Problem 3, 4, 5 respectively. We argue that the search and decisional versions of these problems have similar hardness.

The message security for BIKE-1 and BIKE-2 rely on the decisional version of Problem 3 as defined next:

Problem 3a (Decisional parity-$(2,1)$-QCSD)**.**
Instance: *Given* c, h *in* $\mathcal{R}$*, an integer* $t > 0$*,* $|h|$ *odd and* $|c| + t$ *even.*
Property: *Decides if there exist* e_0, e_1 *in* $\mathcal{R}$ *such that* $|e_0|+|e_1| = t$ *and* $e_0+e_1h = c$*.*

There are two differences between the search problem given in Problem 3 and its decisional version given in Problem 3a. One is a parity condition on the instance, and the other is the equality for the error weight restriction instead of inequality. Using the inequality is a common practice in coding theory and corresponds to a situation where one wishes to decode up to a bound. In fact, this problem was written in this way in the Berlekamp, McEliece and Von-Tilborg's seminal paper [6]. The two problems (differing on whether the equality or inequality is used) are closely related and are essentially of same difficulty. Note that in the same seminal paper, the codeword finding problem is described with an equality.

The other difference concerns the parity property. The parity of a sum (respectively product) is equal to the sum (respectively product) of the parities – this comes directly from the quasi-cyclicity and the underlying polynomial ring structure. The weight of h is odd because, by construction, public keys have an odd weight. Consequently, s and (e_1, e_2) must have the same parity else the property is trivially false. Another variant of Problem 3 is needed for the key indistinguishability of BIKE-3 (note that w is even and $w/2$ is odd).

Problem 3b (Decisional balanced-$(2,1)$-QCSD)**.** *(w even, $w/2$ odd)*
Instance: *Given* f_0, f_1 *in* $\mathcal{R}$*, an integer* $w > 0$*,* $|f_1|$ *odd and* $|f_0|$ *even.*
Property: *Decides whether there exist* h_0, h_1 *in* $\mathcal{R}$ *such that* $|h_0| = |h_1| = w/2$ *and* $h_1 + h_0f_1 = f_0$*.*

Here, in addition, there is a balancedness constraint: both halves of the solution (h_0, h_1) must have the same weight. In practice, a proportion $\Theta(1/\sqrt{w})$ of pairs (h_0, h_1) of total weight w are balanced and the problem cannot become significantly easier on average in the balanced case.

Similarly the key indistinguishability for BIKE-1 and BIKE-2 requires a balanced variant of Problem 4.

Problem 4a (Decisional balanced-$(2, 1)$-QCCF). *(w even, $w/2$ odd)*
Instance: *Given h in $\mathcal{R}$, $|h|$ odd, an integer $w > 0$.*
Property: *Decides if there exist h_0, h_1 in $\mathcal{R}$ such that $|h_0| = |h_1| = w/2$ and $h_0 + h_1 h = 0$.*

Finally, for the message security of BIKE-3 we need a decisional balanced variant of Problem 5

Problem 5a (Decisional balanced-$(3, 1)$-QCSD). *(t even)*
Instance: *Given s_0, s_1, f_0, f_1 in $\mathcal{R}$, an integer $t > 0$, $|f_1|$ odd, $|f_0|, |s_1|, |s_0| + t/2$ even.*
Property: *Decide if there exist e, e_0, e_1 in $\mathcal{R}$ such that $|e| = t/2$, $|e_0| + |e_1| = t$, $e + e_1 f_0 = s_0$ and $e_0 + e_1 f_1 = s_1$.*

Again here, the probability for a triple (e, e_0, e_1) to be balanced with $|e| = t/2$ and $|e_0| + |e_1| = t$ is $\Theta(1/\sqrt{t})$.

To conclude our remarks, we reiterate that none of the variations described above have a significant impact on the hardness of the problems. The parity issue is purely technical. In fact, for given system parameters, the parity of many objects appearing in the protocol is imposed. We need to impose the same parity in the sequence of games or we could obtain a trivial (but meaningless) distinguisher. On the other hand, the matter of balancedness could in principle affect the hardness of the problem, but in practice the impact is very limited. This is because balanced words appear with polynomial probability, and thus the balanced problems cannot be fundamentally easier than generic ones. In light of these considerations, we can simply refer to the generic problems, both in the statement of Theorem 2 and in its proof.

Remark 2. *In the context of the general syndrome decoding problem, there is a search to decision reduction. For the quasi-cyclic case, no such reduction is known, however the best known attacks for the decisional case correspond to the search case.*

Proof of Theorem 2. To begin, note that we model the hash function $\mathbf{K}$ as a random oracle. We will use a sequence of games with the goal of showing that an adversary distinguishing one game from another can be exploited to break one or more of the problems cited above in polynomial time (see Section 5.1 for definitions).

First let us instantiate the IND-CPA game for all three BIKE variants. For all variants, the game will use the following randomness

$$\begin{cases} m & \overset{\$}{\leftarrow} & \mathcal{R} & \\ g & \overset{\$}{\leftarrow} & \mathcal{R} & |g| \text{ odd } (\textit{i.e. } g \text{ invertible}) \\ (h_0, h_1) & \overset{\$}{\leftarrow} & \mathcal{R}^2 & |h_0| = |h_1| = w/2 \text{ odd} \\ (e_0, e_1) & \overset{\$}{\leftarrow} & \mathcal{R}^2 & |e_0| + |e_1| = t \\ e & \overset{\$}{\leftarrow} & \mathcal{R} & |e| = t/2 \end{cases}$$

The output (sk, pk) of $\textsc{Gen}(\lambda)$ will be $sk = (h_0, h_1)$ for all variants and

$$\begin{cases} pk = (f_0, f_1) = (gh_1, gh_0) & \text{for BIKE-1,} \\ pk = h = h_1 h_0^{-1} & \text{for BIKE-2,} \\ pk = (f_0, f_1) = (h_1 + gh_0, g) & \text{for BIKE-3.} \end{cases}$$

For both valid and random pk, the output (c, K) of $\textsc{Encaps}(pk)$ will be $K = \mathbf{K}(e_0, e_1) \in \mathcal{K} = \{0,1\}^{\ell_K}$ for all variants and

$$\begin{cases} c = (mf_0 + e_0, mf_1 + e_1) & \text{with } pk = (f_0, f_1),\ |f_0| \text{ and } |f_1| \text{ odd,} & \text{for BIKE-1,} \\ c = e_0 + e_1 h & \text{with } pk = h,\ |h| \text{ odd,} & \text{for BIKE-2,} \\ c = (e + e_1 f_0, e_0 + e_1 f_1) & \text{with } pk = (f_0, f_1),\ |f_0| \text{ even, } |f_1| \text{ odd,} & \text{for BIKE-3.} \end{cases}$$

Let $\mathcal{A}$ be a probabilistic polynomial-time adversary playing the IND-CPA game against our scheme, and consider the following games.

Game G_1: This corresponds to an honest run of the protocol, and is the same as the original IND-CPA game. In particular, the simulator has access to all keys and randomness.

Game G_2: In this game, the goal is to forget the secret key, and to generate a random public key. It is the same as the previous game where step 1: is replaced by

$$\begin{cases} 1: & pk = (f_0, f_1) \overset{\$}{\leftarrow} \mathcal{R}^2, & |f_0|, |f_1| \text{ odd}, & \text{for BIKE-1} \\ 1: & pk = h \overset{\$}{\leftarrow} \mathcal{R}, & |h| \text{ odd}, & \text{for BIKE-2} \\ 1: & pk = (f_0, f_1) \overset{\$}{\leftarrow} \mathcal{R}^2, & |f_0| \text{ even}, |f_1| \text{ odd}, & \text{for BIKE-3} \end{cases}$$

An adversary distinguishing between these two games is therefore able to distinguish between a well-formed public key and a randomly-generated one (of suitable parity). To distinguish $\boldsymbol{G}_1$ from $\boldsymbol{G}_2$ the adversary must:

BIKE-1: Distinguish (gh_1, gh_0) from a random pair of invertible (*i.e.* odd weight) elements of $\mathcal{R}$. First, for any $(h_0, h_1) \in \mathcal{R}^2$ the distributions

- (gh_1, gh_0) with $g \overset{\$}{\leftarrow} \mathcal{R}$, $|g|$ odd, and
- $(g'h_1h_0^{-1}, g')$ with $g' \overset{\$}{\leftarrow} \mathcal{R}$, $|g'|$ odd

are identical. If the latter can be distinguished from random then $h_1h_0^{-1}$ can be distinguished from random. Thus, for $\boldsymbol{G}_1$ BIKE-1, $\mathrm{Adv}^{\mathbf{G}_1}(\mathcal{A}) \leq \mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) + \mathrm{Adv}^{(2,1)-\mathrm{QCCF}}(\mathcal{A}')$.

BIKE-2: Distinguish $h_1h_0^{-1}$ from a random invertible element of $\mathcal{R}$. And so, once again: $\mathrm{Adv}^{\mathbf{G}_1}(\mathcal{A}) \leq \mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) + \mathrm{Adv}^{(2,1)-\mathrm{QCCF}}(\mathcal{A}')$.

BIKE-3: Distinguish $(h_0 + h_1g, g)$ from a random pair of elements of $\mathcal{R}$ with (even,odd) weight. And so, $\mathrm{Adv}^{\mathbf{G}_1}(\mathcal{A}) \leq \mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) + \mathrm{Adv}^{(2,1)-\mathrm{QCSD}}(\mathcal{A}')$.

Thus we have respectively:

$$\mathrm{Adv}^{\mathbf{G}_1}(\mathcal{A}) - \mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) \leq \mathrm{Adv}^{(2,1)\text{-QCCF}}(\mathcal{A}') \text{ for BIKE-1 and BIKE-2}$$

and

$$\mathrm{Adv}^{\mathbf{G}_1}(\mathcal{A}) - \mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) \leq \mathrm{Adv}^{(2,1)\text{-QCSD}}(\mathcal{A}') \text{ for BIKE-3}$$

where $\mathcal{A}'$ is a polynomial-time adversary for the underlying problem.

Game $\boldsymbol{G}_3$: Now, the simulator also picks a random ciphertext. Thus the game is the same as $\boldsymbol{G}_2$, but we replace step 4: by

$$\begin{cases} 4: & c^* = (c_0^*, c_1^*) \overset{\$}{\leftarrow} \mathcal{R}^2, & |c_0^*| + |c_1^*| + t \text{ even}, & \text{for BIKE-1} \\ 4: & c^* \overset{\$}{\leftarrow} \mathcal{R}, & |c^*| \text{ odd}, & \text{for BIKE-2} \\ 4: & c^* = (c_0^*, c_1^*) \overset{\$}{\leftarrow} \mathcal{R}^2, & |c_1^*| + t, |c_0^*| + t/2 \text{ even}, & \text{for BIKE-3} \end{cases}$$

An adversary distinguishing between these two games is therefore able to distinguish between a well-formed ciphertext and a randomly-generated one (of suitable parity). To distinguish $\boldsymbol{G}_2$ from $\boldsymbol{G}_3$ the adversary must:

BIKE-1: Given random (f_0, f_1) in $\mathcal{R}$ of odd weight, distinguish a noisy codeword $(c_0, c_1) = (mf_0 + e_0, mf_1 + e_1)$ from a random word (c_0^*, c_1^*) of identical parity. When m is uniformly distributed this is not different from distinguishing the syndrome of (c_0, c_1) using the parity check[4] (f_1^T, f_0^T), that is $e_0f_1 + e_1f_0$, from a random element of $\mathcal{R}$ of identical parity. This won't be different from distinguishing $e_0 + e_1f_0{f_1}^{-1}$ from a random element of $\mathcal{R}$ of identical parity. Because f_0 and f_1 are random, we then have: $\mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) \leq \mathrm{Adv}^{\mathbf{G}_3}(\mathcal{A}) + \mathrm{Adv}^{(2,1)-\mathrm{QCSD}}(\mathcal{A}'')$.

BIKE-2: Given h random in $\mathcal{R}$ of odd weight, distinguish $e_0 + e_1h$ from a random element of $\mathcal{R}$ of identical parity. And so, once again, $\mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) \leq \mathrm{Adv}^{\mathbf{G}_3}(\mathcal{A}) + \mathrm{Adv}^{(2,1)-\mathrm{QCSD}}(\mathcal{A}'')$.

BIKE-3: Given random (f_0, f_1) in $\mathcal{R}$ of (even,odd) weight, distinguish $(c_0, c_1) = (e + e_1f_0, e_0 + e_1f_1)$ from a random pair (c_0^*, c_1^*) of elements of $\mathcal{R}$. Here c_1^* must have the parity of (e_0, e_1) and c_0^* must have the parity of e. This corresponds to the decisional balanced $(3,1)$-QCSD challenge, and so $\mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) \leq \mathrm{Adv}^{\mathbf{G}_3}(\mathcal{A}) + \mathrm{Adv}^{(3,1)-\mathrm{QCSD}}(\mathcal{A}'')$.

If an adversary is able to distinguish game $\boldsymbol{G}_2$ from game $\boldsymbol{G}_3$, then it can solve one of the QCSD problems. Hence, we have either:

$$\mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) - \mathrm{Adv}^{\mathbf{G}_3}(\mathcal{A}) \leq \mathrm{Adv}^{(2,1)\text{-QCSD}}(\mathcal{A}'') \text{ for BIKE-1 and BIKE-2}$$

and $\mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) - \mathrm{Adv}^{\mathbf{G}_3}(\mathcal{A}) \leq \mathrm{Adv}^{(3,1)\text{-QCSD}}(\mathcal{A}'')$ for BIKE-3 where $\mathcal{A}''$ is a polynomial-time adversary for the underlying problem.

Note that at this point, the adversary receives only random values for public key and ciphertext, and is called to distinguish between K_0 and K_1. Now, the latter is generated uniformly at random, while the former is pseudorandom (since $\mathbf{K}$ is modeled as a random oracle), and therefore the adversary only has negligible advantage, say ϵ. So in the end, we have:

$$\mathrm{Adv}^{\text{IND-CPA}}(\mathcal{A}) \leq \mathrm{Adv}^{(2,1)\text{-QCCF}}(\mathcal{A}') + \mathrm{Adv}^{(2,1)\text{-QCSD}}(\mathcal{A}'') + \epsilon. \tag{6}$$

for BIKE-1 and BIKE-2 or

$$\mathrm{Adv}^{\text{IND-CPA}}(\mathcal{A}) \leq \mathrm{Adv}^{(2,1)\text{-QCSD}}(\mathcal{A}') + \mathrm{Adv}^{(3,1)\text{-QCSD}}(\mathcal{A}'') + \epsilon. \tag{7}$$

for BIKE-3.

□

[4] Indeed $(f_0, f_1) \cdot (f_1^T, f_0^T)^T = f_0f_1 + f_1f_0 = 0$

6.2 IND-CCA Security

As we mentioned in Section 2.2, the IND-CCA secure version of each BIKE variant is obtained via a specific conversion which transforms the underlying encryption scheme into an IND-CCA secure KEM. Therefore, the security statements will be based on the OW-CPA security of the underlying cryptosystems. In particular, due to the differences in the nature of these cryptosystems, the choice of conversion is not always the same, and we will discuss the three variants individually as promised. All conversions are taken from [24], and so are the corresponding security results in the Random Oracle Model (ROM). However, in the Quantum Random Oracle Model (QROM), no proofs are given in [24]. Instead, the authors propose alternative solutions which are very loose and present some disadvantages, such as requiring an additional hash value to be appended to the ciphertext. This gap was filled in [25], where proofs (in the QROM) are given for all the conversions of [24], and successively improved in [26] for some particular cases. Therefore, we are able to present a completely detailed scenario for the chosen conversions, both in the ROM and in the QROM. Note that, in all cases, the resulting KEMs have the exact same DFR as the underlying cryptosystem: we will denote this by ρ.

6.2.1 BIKE-1-CCA

In this case, the underlying cryptosystem is McEliece. To obtain an IND-CCA secure KEM, we apply the transformation called $\mathsf{FO}^{\not\perp}$ from [24]. This can be decomposed in two parts: the first part T makes the cryptosystem deterministic by computing the randomness as the hash output of $\mathbf{G}$, while the second part $\mathsf{U}^{\not\perp}$ converts such a deterministic cryptosystem into a KEM, using re-encryption and implicit rejection to guarantee IND-CCA security. We have the following result.

Theorem 3. *Let $\mathcal{A}$ be an IND-CCA adversary for BIKE-1-CCA in the ROM, running in time θ and issuing at most q_D decapsulation queries and q_G+q_K random oracle queries (respectively to the random oracles $\mathbf{G}$ and $\mathbf{K}$). Then there exists a OW-CPA adversary $\mathcal{A}'$ for QC-MDPC McEliece, running in approximately the same time, such that*

$$\mathrm{Adv}_{KEM}^{IND-CCA}(\mathcal{A}) \leq q_G \cdot \rho + \frac{q_K}{\binom{n}{t}} + (q_G+1) \cdot \mathrm{Adv}_{PKE}^{OW-CPA}(\mathcal{A}'). \qquad (8)$$

A proof of Theorem 3 can be easily obtained by combining those of Theorems 3.1 and 3.4 of [24]. The term $\binom{n}{t}$ corresponds to the size of the message space, i.e. the set of vectors $(e_0, e_1) \in \mathcal{R}^2$ of combined weight t. Note that the probabilistic nature of the underlying cryptosystem, and the consequent usage of the random oracle $\mathbf{G}$, is a source of looseness (appearing in the terms containing q_G).

In the QROM, we have the following result from [25].

Theorem 4. *Let $\mathcal{A}$ be an IND-CCA adversary for BIKE-1-CCA in the QROM, running in time θ and issuing at most q_D decapsulation queries and q_G+q_K random oracle queries (respectively to the random oracles $\mathbf{G}$ and $\mathbf{K}$). Then there exists a OW-CPA adversary $\mathcal{A}'$ for QC-MDPC McEliece, running in approximately the same time, such that*

$$\mathrm{Adv}_{KEM}^{IND-CCA}(\mathcal{A}) \leq 4q_G \cdot \sqrt{\rho} + \frac{2q_K}{\sqrt{\binom{n}{t}}} + 2(q_G + q_K) \cdot \sqrt{\mathrm{Adv}_{PKE}^{OW-CPA}(\mathcal{A}')}. \quad (9)$$

Theorem 4 was proved in [25, Theorem 1], where again we substitute $\binom{n}{t}$ for the size of the message space. Note that this reduction is even looser than the previous one, partly due to, again, the probabilistic nature of McEliece, and partly because of the inherent difficulties connected to proofs in the QROM.

6.2.2 BIKE-2-CCA

In this case, the underlying cryptosystem is Niederreiter. The deterministic nature of this cryptosystem will be advantageous. In fact, it is not necessary to "derandomize" the scheme, and to obtain an IND-CCA secure KEM, it is enough to apply just $\mathsf{U}^{\not\perp}$. We have the following result.

Theorem 5. *Let $\mathcal{A}$ be an IND-CCA adversary for BIKE-2-CCA in the ROM, running in time θ and issuing at most q_D decapsulation queries and q_K random oracle queries to the random oracle $\mathbf{K}$. Then there exists a OW-CPA adversary $\mathcal{A}'$ for QC-MDPC Niederreiter, running in approximately the same time, such that*

$$\mathrm{Adv}_{KEM}^{IND-CCA}(\mathcal{A}) \leq 2\rho + \frac{q_K}{\binom{n}{t}} + \mathrm{Adv}_{PKE}^{OW-CPA}(\mathcal{A}'). \quad (10)$$

A proof of Theorem 5 follows immediately from that of [24, Theorem 3.1], with a little addition. In fact, the term 2ρ is obtained when game-hopping from OW-CPA to an intermediate notion known as OW-PCA (which stands for Plaintext Checking) and effectively replaces the derandomization step. In doing so, we have eliminated the need to apply T, and the random oracle $\mathbf{G}$, and consequently a large source of looseness, allowing us to obtain a very tight conversion.

In the QROM, we have the following result from [25].

Theorem 6. *Let $\mathcal{A}$ be an IND-CCA adversary for BIKE-2-CCA in the QROM, running in time θ and issuing at most q_D decapsulation queries and q_K random oracle queries to the random oracle $\mathbf{K}$. Then there exists a OW-CPA adversary $\mathcal{A}'$ for QC-MDPC Niederreiter, running in approximately the same time, such that*

$$\mathrm{Adv}_{KEM}^{IND-CCA}(\mathcal{A}) \leq \frac{2q_K}{\sqrt{\binom{n}{t}}} + 2\sqrt{(q_K+1)(\mathrm{Adv}_{PKE}^{OW-CPA}(\mathcal{A}') + 4\rho)}. \tag{11}$$

For Theorem 6, we used the improved technique detailed in [26, Theorem 6], where, as above, we have to include an additional 2ρ term to make up for the lack of perfect correctness in passing from OW-CPA to OW-qPCA. This reduction is obviously much tighter than its equivalent for BIKE-1-CCA.

6.2.3 BIKE-3-CCA

This case is similar to BIKE-1-CCA, in that the underlying cryptosystem is again probabilistic, namely Ouroboros. The conversion applied is exactly the same as for BIKE-1-CCA, and therefore the resulting bounds are also the same as those of Theorems 3 and 4. We do not report them to avoid unnecessary repetition.

6.3 Public Keys and Subcodes

In this section, we prove that one can efficiently sample an *invertible* element from $\mathbb{F}_2[X]/\langle X^r - 1\rangle$ by taking any polynomial $h \overset{\$}{\leftarrow} \mathbb{F}_2[X]/\langle X^r - 1\rangle$ such that $|h|$ is odd. If this element was not invertible, the public code produced in BIKE-1 and BIKE-3 would be a subcode of the private one.

Lemma 1. *Let $h \in \mathbb{F}_2[X]$ have even weight. Then h is not invertible modulo $X^r - 1$.*

Proof. We show that $(X-1) \mid h$ by induction on $|h|$. For $|h| = 0$ trivially $(X-1) \mid h$. Assume that $(X-1) \mid h$ whenever $|h| = 2k$ for some $k \geqslant 0$. Now consider any $h \in \mathbb{F}_2[X]$ with weight $|h| = 2(k+1)$, and take two distinct terms X^i, X^j of h such that $i < j$. Define $h' = h - X^i - X^j$, so that $|h'| = 2k$. Then $(X-1) \mid h'$ by induction, i.e. $h' = (X-1)h''$ for some $h'' \in \mathbb{F}_2[X]$. Hence $h = h' + X^i + X^j = (X-1)h'' + X^i(X^{j-i}+1) = (X-1)h'' + X^i(X-1)(X^{j-i-1} + \cdots + 1) = (X-1)(h'' + X^i(X^{j-i-1} + \cdots + 1))$, and therefore $(X-1) \mid h$. □

Theorem 7. *Let r a prime such that $(X^r-1)/(X-1) \in \mathbb{F}_2[X]$ is irreducible. Then any $h \in \mathbb{F}_2[X]$ with $\deg(h) < r$ is invertible modulo $X^r - 1$ iff $h \neq X^{r-1} + \cdots + 1$ and $|h|$ is odd.*

Proof. Take a term X^i of h. Then $\left|h + X^i\right| = |h| - 1$ is even, and by Lemma 1 $(X - 1) \mid (h + X^i)$. Hence $h \bmod (X - 1) = X^i \bmod (X - 1) = 1$, meaning that h is invertible modulo $X - 1$.

Now, since $(X^r - 1)/(X - 1) = X^{r-1} + \cdots + 1$ is irreducible, if $\deg(h) < r - 1$ then $\gcd(h, X^{r-1} + \cdots + 1) = 1$, and if $\deg(h) = r - 1$, then $\gcd(h, X^{r-1} + \cdots + 1) = \gcd(h + X^{r-1} + \cdots + 1, X^{r-1} + \cdots + 1) = 1$, since $\deg(h + X^{r-1} + \cdots + 1) < r - 1$. Hence h is invertible modulo $X^{r-1} + \cdots + 1$.

Therefore, the combination of the inverses of h modulo $X - 1$ and modulo $X^{r-1} + \cdots + 1$ via the Chinese remainder theorem is well defined, and by construction it is the inverse of h modulo $(X - 1)(X^{r-1} + \cdots + 1) = X^r - 1$. □

Corollary 1. *One can efficiently sample an invertible element from* $\mathbb{F}_2[X]/\langle X^r - 1\rangle$ *by taking any polynomial* $h \xleftarrow{\$} \mathbb{F}_2[X]/\langle X^r - 1\rangle$ *such that* $|h|$ *is odd.*

7 Advantages and Limitations (2.B.6)

This document presents BIKE, a suite of key encapsulation mechanisms (KEM) composed by three IND-CPA secure variants, called BIKE-1, BIKE-2 and BIKE-3, and three IND-CCA variants, called BIKE-1-CCA, BIKE-2-CCA and BIKE-3-CCA. Each variant has its own pros and cons, which we will illustrate below.

All BIKE variants are tied together by the fact that they are based on quasi-cyclic moderate density parity-check (QC-MDPC codes), which can be efficiently decoded through bit flipping decoding techniques. This kind of decoder is extremely simple: it estimates what are the positions most likely in error, flip them and observes whether the result is better (smaller syndrome weight) than before or not. This process converges very quickly; in particular, Section 2.4.2 presents a 1-iteration bit flipping decoder.

The three IND-CPA secure BIKE variants presented in Round 1 were designed to use ephemeral keys. The main reason for this choice, is that it inherently defeats the GJS reaction attack mentioned in Section 5, which needs to observe a large number of decodings for the same private key (something impossible when ephemeral keys are used). As a consequence, key generation must be efficient, since it is executed at every key encapsulation. Previous works based on QC-MDPC codes compute a polynomial inversion operation in order to obtain a QC-MDPC public key in systematic form. The polynomial inversion is an expensive operation. BIKE-1 completely avoids the polynomial inversion by not relying on public keys in systematic form. Instead, it hides the private sparse structure by multiplying it by a dense polynomial of odd weight sampled uniformly at random. This leads to an increased public key size but results in a very efficient key generation process (it becomes the fastest process among key generation, encapsulation and decapsulation operations). BIKE-2 uses public keys in systematic form, but thanks to our batch key generation technique discussed in Section 3.7, the amortized cost can decrease up to 84%, becoming less expensive than the bit flipping decoder. BIKE-3 has the advantage of relying on a single security assumption, namely Quasi-Cyclic Syndrome Decoding (QCSD), as detailed in Section 6.1. This makes the security reduction simpler. Besides the bit flipping algorithm and the eventual polynomial inversion (only necessary BIKE-2), all other operations in the BIKE suite consist of simple products of binary vectors, an operation that can be easily optimized for all sorts of hardware and software applications.

Regarding communication bandwidth, in BIKE-1 and BIKE-3 all public keys, private keys and cryptograms are n bits long, corresponding to the sizes of the messages exchanged by the parties. BIKE-2 offers smaller public keys and ciphertexts, r bits only, corresponding to the sizes of the messages exchanged by the parties. Two messages of same size (either n or r bits) are exchanged per key encapsulation. In practice, these numbers range from 1.24 KB per message (BIKE-2 security level 1), up to 8.82 KB per message (BIKE-3 security level 5), in the IND-CPA setting. These numbers seem fairly reasonable when compared to the the average size of a website page (currently near 2MB [2]), just as an example.

Regarding security, all BIKE variants rely their security on very well-known coding-theory problems: quasi-cyclic syndrome decoding and quasi-cyclic codeword finding problems (with BIKE-3 relying only on the former as we just mentioned). The best strategies to solve these problems are based on Information Set Decoding (ISD) techniques, a research field that has a very long history (Prange's seminal work dates back 1962) and which has seem very little improvement along the years. Moreover, we show that in the quantum setting, Grover's algorithm used on top of the seminal Prange ISD algorithm is still the most preferable choice in our case.

One point worth noting is that the bit flipping decoding techniques used in the original BIKE proposal come with a relatively high decoding failure rate. While this is not a problem in terms of reaction attacks (thanks to the ephemeral usage of the key pairs), this would prevent the original BIKE variants from achieving higher security notions such as IND-CCA. For Round 2 submission, we devised an improved decoding techniques, called the Backflip decoder. It is presented in Section 2.4.3 and it attains negligible decoding failure rates. These can be in fact modulated to reach exactly the desired threshold (e.g. 2^{-128} for Security Level 1), at the cost of a minor increase in block size.

Thanks to the improved decoder, it was possible to design IND-CCA secure versions of the three BIKE variants, which were therefore named BIKE-1-CCA, BIKE-2-CCA and BIKE-3-CCA respectively. Security for these versions is obtained via a combination of various generic conversions such as [24, 25, 26]. The three BIKE-CCA versions present a slightly higher cost in term of performance, in exchange for the higher security level and the ability to use static keys. Note that in this scenario, the flexibility of having three BIKE variants is even more evident, as the roles shift compared to the previous case. While BIKE-1 shines as an ephemeral key encapsulation mechanism, largely thanks to the simple description and the fast, inversion-less key generation, BIKE-2 seems like a more natural choice for static keys. In fact, when key generation is performed only once the disadvantage of the slower algorithm is less relevant, while on the other hand halving the size of ciphertext and public key is beneficial from a communciation bandwidth perspective when many key exchanges are performed. Moreover, the deterministic nature of the underlying cryptosystem (which for BIKE-2 is Niederreiter) allows for a tighter and simpler security reduction.

Finally, regarding intellectual property, to the best of our knowledge, the IND-CPA secure BIKE-1 and BIKE-2 are not covered by any patent. BIKE-3 is covered by a patent whose owners are willing to grant a non-exclusive license for the purpose of implementing the standard *without compensation* and under reasonable terms and conditions that are demonstrably free of any unfair discrimination, as denoted in the accompanying signed statements. We emphasize that BIKE-1 and BIKE-2 are *not covered* by the aforementioned patent, and that the BIKE team is willing to drop BIKE-3 if this ever becomes a disadvantage when comparing our suite with other proposals. The rationale above focused on the IND-CPA variants equally applies to the IND-CCA variants.

Overall, taking all these considerations into account, we believe that BIKE is a well-rounded and promising candidate for post-quantum key exchange standardization.

8 Acknowledgments

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). Shay Gueron was also partially supported by the Israel Science Foundation (grant No. 1018/16). Paulo S. L. M. Barreto was partially supported by Intel and FAPESP through the project "Efficient Post-Quantum Cryptography for Building Advanced Security Applications" (grant No. 2015/50520-6). Tim Güneysu was partially supported by Intel (http://www.icri-cars.org). We thank Danilo Gligoroski for his comments in the NIST mailing list. The logo presented in the cover page was designed by Szilard Nagy. The reference code was developed by Nir Druker, Shay Gueron, Rafael Misoczki, Tim Güneysu, Tobias Oder and Slim Bettaieb.

References

[1] Michael Alekhnovich. More on average case vs approximation complexity. In *FOCS 2003*, pages 298–307. IEEE, 2003.

[2] HTTP Archive. Http archive report, 2017. `http://httparchive.org/trends.php`.

[3] Elaine B Barker and John Michael Kelsey. *Recommendation for random number generation using deterministic random bit generators (revised)*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory, 2012.

[4] Paulo S. L. M. Barreto, Shay Gueron, Tim Guneysu, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, and Jean-Pierre Tillich. CAKE: Code-based Algorithm for Key Encapsulation. Cryptology ePrint Archive, Report 2017/757, 2017. `https://eprint.iacr.org/2017/757.pdf`. To appear in the 16th IMA International Conference on Cryptography and Coding.

[5] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How 1+1=0 improves information set decoding. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, 2012.

[6] Elwyn Berlekamp, Robert J. McEliece, and Henk van Tilborg. On the inherent intractability of certain coding problems (corresp.). *Information Theory, IEEE Transactions on*, 24(3):384 – 386, may 1978.

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

[8] Céline Blondeau, Benoît Gérard, and Jean-Pierre Tillich. Accurate estimates of the data complexity and success probability for various cryptanalyses. *Des. Codes Cryptogr.*, 59(1-3):3–34, 2011.

[9] Pierre-Louis Cayrel, Gerhard Hoffmann, and Edoardo Persichetti. Efficient implementation of a cca2-secure variant of McEliece using generalized Srivastava codes. In *Proceedings of PKC 2012, LNCS 7293, Springer-Verlag*, pages 138–155, 2012.

[10] Julia Chaulet. *Étude de cryptosystèmes à clé publique basés sur les codes MDPC quasi-cycliques.* Thèse de doctorat, University Pierre et Marie Curie, March 2017.

[11] 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.

[12] 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.

[13] Thomas M. Cover and Joy A. Thomas. *Information Theory.* Wiley Series in Telecommunications. Wiley, 1991.

[14] 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.

[15] Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In Tanja Lange and Tsuyoshi Takagi, editors, *PQCrypto 2017*, volume 10346 of *LNCS*, pages 18–34. Springer, 2017.

[16] Nir Drucker and Shay Gueron. A toolbox for software optimization of qc-mdpc code-based cryptosystems. Cryptology ePrint Archive, December 2017. `http://eprint.iacr.org/`.

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

[18] Shay Gueron. A j-lanes tree hashing mode and j-lanes SHA-256. *Journal of Information Security*, 4(01):7, 2013.

[19] Shay Gueron. Parallelized hashing via j-lanes and j-pointers tree modes, with applications to SHA-256. *Journal of Information Security*, 5(03):91, 2014.

[20] Shay Gueron. A-toolbox-for-software-optimization-of-qc-mdpc-code-based-cryptosystems, 2017. `https://github.com/Shay-Gueron/A-toolbox-for-software-optimization-of-QC-MDPC-code-based-cryptosystems`.

[21] Shay Gueron and Vlad Krasnov. Simultaneous hashing of multiple messages. *Journal of Information Security*, 3(04):319, 2012.

[22] 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.

[23] 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`.

[24] 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.

[25] Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. IND-CCA-secure key encapsulation mechanism in the Quantum Random Oracle Model, revisited. In *Annual International Cryptology Conference*, pages 96–125. Springer, 2018.

[26] Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. Tighter security proofs for generic key encapsulation mechanism in the Quantum Random Oracle Model. Cryptology ePrint Archive, Report 2019/134, 2019. `http://eprint.iacr.org/2019/134`.

[27] 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.

[28] Gil Kalai and Nathan Linial. On the distance distribution of codes. *IEEE Trans. Inform. Theory*, 41(5):1467–1472, September 1995.

[29] Gianluigi Liva and Hannes Bartz. Protograph-based quasi-cyclic MDPC codes for mceliece cryptosystems. *CoRR*, abs/1801.07484, 2018.

[30] 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.

[31] Florence J. MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes*. North–Holland, Amsterdam, fifth edition, 1986.

[32] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT'93*, volume 765 of *LNCS*, pages 386–397, Lofthus, Norway, May 1993. Springer.

[33] Ingo Von Maurich, Tobias Oder, and Tim Güneysu. Implementing qc-mdpc mceliece encryption. *ACM Trans. Embed. Comput. Syst.*, 14(3):44:1–44:27, April 2015.

[34] 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.

[35] Daniele Micciancio. Improving lattice based cryptosystems using the hermite normal form. *Cryptography and lattices*, pages 126–145, 2001.

[36] 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.

[37] Peter L Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[38] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions*, IT-8:S5–S9, 1962.

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

[40] Nicolas Sendrier and Valentin Vasseur. On the decoding failure rate of qc-mdpc bit-flipping decoders. Cryptology ePrint Archive, Report 2018/1207, 2018. `https://eprint.iacr.org/2018/1207` - To appear in PQCrypto 2019.

[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] Ingo Von Maurich and Tim Güneysu. Lightweight code-based cryptography: Qc-mdpc mceliece encryption on reconfigurable devices. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2014.

[43] Christof Zalka. Grover's quantum searching algorithm is optimal. *Phys. Rev. A*, 60:2746–2751, October 1999.

A Proof of Theorem 1

Let us recall the theorem we want to prove.

Theorem 1. *Under assumption 1, the probability P_{err} that the bit flipping algorithm fails to decode with fixed threshold $\tau = \frac{1}{2}$ is upper-bounded by*

$$P_{err} \leq \frac{1}{\sqrt{\alpha \pi t}} e^{\frac{\alpha t w}{8} \ln\left(1-\varepsilon^2\right) + \frac{\alpha t}{8} \ln(n) + O(t)},$$

where $\varepsilon \overset{def}{=} e^{-\frac{2wt}{n}}$.

We will denote in the whole section by $h(x)$ the entropy (in nats) of a Bernoulli random variable of parameter x, that is $h(x) \overset{\text{def}}{=} -x \ln x - (1-x)\ln(1-x)$.

A.1 Basic tools

A particular quantity will play a fundamental role here, the Kullback-Leibler divergence (see for instance [13])

Definition 7. Kullback-Leibler divergence
Consider two discrete probability distributions $\mathbf{p}$ *and* $\mathbf{q}$ *defined over a same discrete space $\mathcal{X}$. The Kullback-Leibler divergence between* $\mathbf{p}$ *and* $\mathbf{q}$ *is defined by*

$$D(\mathbf{p} \| \mathbf{q}) = \sum_{x \in \mathcal{X}} p(x) \ln \frac{p(x)}{q(x)}.$$

We overload this notation by defining for two Bernoulli distributions $\mathcal{B}(p)$ and $\mathcal{B}(q)$ of respective parameters p and q

$$D(p \| q) \overset{def}{=} D(\mathcal{B}(p) \| \mathcal{B}(q)) = p \ln\left(\frac{p}{q}\right) + (1-p)\ln\left(\frac{1-p}{1-q}\right).$$

We use the convention (based on continuity arguments) that $0 \ln \frac{0}{p} = 0$ and $p \ln \frac{p}{0} = \infty$.

We will need the following approximations/results of the Kullback-Leibler divergence

Lemma 2. *For any $\delta \in (-1/2, 1/2)$ we have*

$$D\left(\frac{1}{2} \middle\| \frac{1}{2} + \delta\right) = -\frac{1}{2}\ln(1-4\delta^2). \tag{12}$$

For constant $\alpha \in (0,1)$ and δ going to 0 by staying positive, we have

$$D(\alpha\|\delta) = -h(\alpha) - \alpha \ln \delta + O(\delta). \tag{13}$$

For $0 < y < x$ and x going to 0 we have

$$D(x\|y) = x \ln \frac{x}{y} + x - y + O\left(x^2\right). \tag{14}$$

Proof. Let us first prove (12).

$$\begin{aligned}
D\left(\frac{1}{2}\middle\|\frac{1}{2}+\delta\right) &= \frac{1}{2}\ln\frac{1/2}{1/2+\delta} + \frac{1}{2}\ln\frac{1/2}{1/2-\delta} \\
\mathbb{P} &= -\frac{1}{2}\ln(1+2\delta) - \frac{1}{2}\ln(1-2\delta) \\
&= -\frac{1}{2}\ln(1-4\delta^2).
\end{aligned}$$

To prove (13) we observe that

$$\begin{aligned}
D(\alpha\|\delta) &= \alpha \ln\left(\frac{\alpha}{\delta}\right) + (1-\alpha)\ln\left(\frac{1-\alpha}{1-\delta}\right) \\
&= -h(\alpha) - \alpha\ln\delta - (1-\alpha)\ln(1-\delta) \\
&= -h(\alpha) - \alpha\ln\delta + O(\delta).
\end{aligned}$$

For the last estimate we proceed as follows

$$\begin{aligned}
D(x\|y) &= x\ln\frac{x}{y} + (1-x)\ln\frac{1-x}{1-y} \\
&= x\ln\frac{x}{y} - (1-x)\left(-x+y+O\left(x^2\right)\right) \\
&= x\ln\frac{x}{y} + x - y + O\left(x^2\right).
\end{aligned}$$

□

The Kullback-Leibler appears in the computation of large deviation exponents. In our case, we will use the following estimate which is well known and which can be found for instance in [8]

Lemma 3. *Let p be a real number in $(0,1)$ and $X_1, \dots X_n$ be n independent Bernoulli random variables of parameter p. Then, as n tends to infinity:*

$$\mathbb{P}(X_1 + \dots X_n \geq \tau n) = \frac{(1-p)\sqrt{\tau}}{(\tau-p)\sqrt{2\pi n(1-\tau)}} e^{-nD(\tau\|p)}(1+o(1)) \textit{ for } p < \tau < 1, \tag{15}$$

$$\mathbb{P}(X_1 + \dots X_n \leq \tau n) = \frac{p\sqrt{1-\tau}}{(p-\tau)\sqrt{2\pi n\tau}} e^{-nD(\tau\|p)}(1+o(1)) \textit{ for } 0 < \tau < p. \tag{16}$$

A.2 Estimation of the probability that a parity-check equation of weight w gives an incorrect information

A.2.1 Main result

We start our computation by computing the probability that a parity-check equation gives an incorrect information about a bit. We say here that a parity-check equation $\mathbf{h}$ (viewed as a binary word) gives an incorrect information about an error bit e_i that is involved in $\mathbf{h}$ if $\langle \mathbf{h}, \mathbf{e} \rangle \neq e_i$, where $\mathbf{e}$ is the error. This is obtained through the following lemma.

Lemma 4. *Consider a word $\mathbf{h} \in \mathbb{F}_2^n$ of weight w and an error $\mathbf{e} \in \mathbb{F}_2^n$ of weight t chosen uniformly at random. Assume that both w and t are of order $\sqrt{n}$: $w = \Theta(\sqrt{n})$ and $t = \Theta(\sqrt{n})$. We have*

$$\mathbb{P}_{\mathbf{e}}(\langle \mathbf{h}, \mathbf{e} \rangle = 1) = \frac{1}{2} - \frac{1}{2} e^{-\frac{2wt}{n}} \left(1 + O\left(\frac{1}{\sqrt{n}}\right)\right).$$

Remark 3. *Note that this probability is in this case of the same order as the probability taken over errors $\mathbf{e}$ whose coordinates are drawn independently from a Bernoulli distribution of parameter t/n. In such a case, from the piling-up lemma [32] we have*

$$\begin{aligned}\mathbb{P}_{\mathbf{e}}(\langle \mathbf{h}, \mathbf{e} \rangle = 1) &= \frac{1 - \left(1 - \frac{2t}{n}\right)^w}{2} \\ &= \frac{1}{2} - \frac{1}{2} e^{w \ln(1 - 2t/n)} \\ &= \frac{1}{2} - \frac{1}{2} e^{-\frac{2wt}{n}} \left(1 + O\left(\frac{1}{\sqrt{n}}\right)\right).\end{aligned}$$

Let us bring now the following fundamental quantities for $b \in \{0, 1\}$

$$p_b \overset{\text{def}}{=} \mathbb{P}(\langle \mathbf{h}, \mathbf{e} \rangle = 1 | e_1 = b) \tag{17}$$

where without loss of generality we assume that $h_1 = 1$ and $\mathbf{e}$ is an error of weight t and length n chosen uniformly at random.

The proof of this lemma will be done in the following subsection. From this lemma it follows directly that

Corollary 2. *Assume that $w = \Theta(\sqrt{n})$ and $t = \Theta(\sqrt{n})$. Then*

$$p_b = \frac{1}{2} - (-1)^b \varepsilon \left(\frac{1}{2} + O\left(\frac{1}{\sqrt{n}}\right)\right), \tag{18}$$

where $\varepsilon \overset{def}{=} e^{-\frac{2wt}{n}}$.

A.2.2 Proof of Lemma 4

The proof involves properties of the Krawtchouk polynomials. We recall that the (binary) Krawtchouk polynomial of degree i and order n (which is an integer), $P_i^n(X)$ is defined for $i \in \{0, \cdots, n\}$ by:

$$P_i^n(X) \stackrel{\text{def}}{=} \frac{(-1)^i}{2^i} \sum_{j=0}^{i} (-1)^j \binom{X}{j} \binom{n-X}{i-j} \quad \text{where } \binom{X}{j} \stackrel{\text{def}}{=} \frac{1}{j!} X(X-1) \cdots (X-j+1). \tag{19}$$

Notice that it follows on the spot from the definition of a Krawtchouk polynomial that

$$P_k^n(0) = \frac{(-1)^k \binom{n}{k}}{2^k}. \tag{20}$$

Let us define the bias δ by

$$\delta \stackrel{\text{def}}{=} 1 - 2\mathbb{P}_{\mathbf{e}}(\langle \mathbf{h}, \mathbf{e} \rangle = 1).$$

In other words $\mathbb{P}_{\mathbf{e}}(\langle \mathbf{h}, \mathbf{e} \rangle = 1) = \frac{1}{2}(1 - \delta)$. These Krawtchouk polynomials are readily related to δ. We first observe that

$$\mathbb{P}_{\mathbf{e}}(\langle \mathbf{h}, \mathbf{e} \rangle = 1) = \frac{\sum_{\substack{j=1 \\ j \text{ odd}}}^{w} \binom{t}{j} \binom{n-t}{w-j}}{\binom{n}{w}}.$$

Moreover by observing that $\sum_{j=0}^{w} \binom{t}{j} \binom{n-t}{w-j} = \binom{n}{w}$ we can recast the following evaluation of a Krawtchouk polynomial as

$$\begin{aligned}
\frac{(-2)^w}{\binom{n}{w}} P_w^n(t) &= \frac{\sum_{j=0}^{w} (-1)^j \binom{t}{j} \binom{n-t}{w-j}}{\binom{n}{w}} \\
&= \frac{\sum_{\substack{j=0 \\ j \text{ even}}}^{w} \binom{t}{j} \binom{n-t}{w-j} - \sum_{\substack{j=1 \\ j \text{ odd}}}^{w} \binom{t}{j} \binom{n-t}{w-j}}{\binom{n}{w}} \\
&= \frac{\binom{n}{w} - 2 \sum_{\substack{j=1 \\ j \text{ odd}}}^{w} \binom{t}{j} \binom{n-t}{w-j}}{\binom{n}{w}} \\
&= 1 - 2\mathbb{P}_{\mathbf{e}}(\langle \mathbf{h}, \mathbf{e} \rangle = 1) \\
&= \delta.
\end{aligned} \tag{21}$$

To simplify notation we will drop the superscript n in the Krawtchouk polynomial notation. It will be chosen as the length of the MDPC code when will use it in our case. An important lemma that we will need is the following one.

Lemma 5. *For all x in $\{1,\ldots,t\}$, we have*

$$\frac{P_w(x)}{P_w(x-1)} = \left(1+O\left(\frac{1}{n}\right)\right)\frac{n-2w+\sqrt{(n-2w)^2-4w(n-w)}}{2(n-w)}.$$

Proof. This follows essentially from arguments taken in the proof of [31][Lemma 36, §7, Ch. 17]. The result we use appears however more explicitly in [28][Sec. IV] where it is proved that if x is in an interval of the form $\left[0,(1-\alpha)\left(n/2-\sqrt{w(n-w)}\right)\right]$ for some constant $\alpha \in [0,1)$ independent of x, n and w, then

$$\frac{P_w(x+1)}{P_w(x)} = \left(1+O\left(\frac{1}{n}\right)\right)\frac{n-2w+\sqrt{(n-2w)^2-4w(n-w)}}{2(n-w)}.$$

For our choice of t this condition is met for x and the lemma follows immediately. □

We are ready now to prove Lemma 4.

Proof of Lemma 4. We start the proof by using (21) which says that

$$\delta = \frac{(-2)^w}{\binom{n}{w}}P_w^n(t).$$

We then observe that

$$\begin{aligned}
\frac{(-2)^w}{\binom{n}{w}} P_w^n(t) &= \frac{(-2)^w}{\binom{n}{w}} \frac{P_w^n(t)}{P_w^n(t-1)} \frac{P_w^n(t-1)}{P_w^n(t-2)} \cdots \frac{P_w^n(1)}{P_w^n(0)} P_w^n(0) \\
&= \frac{(-2)^w}{\binom{n}{w}} \left(\left(1 + O\left(\frac{1}{n}\right)\right) \frac{n-2w+\sqrt{(n-2w)^2-4w(n-w)}}{2(n-w)} \right)^t P_w^n(0) \text{ (by Lemma 5)} \\
&= \left(1+O\left(\frac{1}{n}\right)\right)^t \left(\frac{n-2w+\sqrt{(n-2w)^2-4w(n-w)}}{2(n-w)} \right)^t \text{ (by (20))} \\
&= e^{t \ln\left(\frac{1-2\omega+\sqrt{(1-2\omega)^2-4\omega(1-\omega)}}{2(1-\omega)}\right)} \left(1+O\left(\frac{t}{n}\right)\right) \text{ where } \omega \overset{\text{def}}{=} \frac{w}{n} \\
&= e^{t \ln\left(\frac{1-2\omega+1-4\omega+O\left(\omega^2\right)}{2(1-\omega)}\right)} \left(1+O\left(\frac{t}{n}\right)\right) \\
&= e^{t \ln\left(\frac{1-3\omega+O\left(\omega^2\right)}{1-\omega}\right)} \left(1+O\left(\frac{t}{n}\right)\right) \\
&= e^{-2t\omega+O\left(\frac{tw^2}{n^2}\right)} \left(1+O\left(\frac{t}{n}\right)\right) \\
&= e^{-\frac{2wt}{n}} \left(1+O\left(\frac{1}{\sqrt{n}}\right)\right),
\end{aligned}$$

where we used at the last equation that $t = \theta(\sqrt{n})$ and $w = \theta(\sqrt{n})$. □

A.3 Estimation of the probability that a bit is incorrectly estimated by the first step of the bit flipping algorithm

We are here in the model where every bit is involved in $w/2$ parity-check equations and each parity-check equation is of weight w. We assume that the bit-flipping algorithm consists in computing for each bit i the syndrome bits corresponding to the parity-checks involving i and taking the majority vote of these syndrome bits. We model each vote of a parity-check by a Bernoulli variable equal to 1 if the information coming from this random variable says that the bit should be flipped. The parameter of this Bernoulli random variable depends on whether or not i is incorrect. When i is correct, then the Bernoulli random variable is of parameter p_0. When i is incorrect, then the Bernoulli random variable is of parameter p_1. We bring in the quantities

$$\begin{aligned} q_0 &\overset{\text{def}}{=} \mathbb{P}(\text{flip the bit}|\text{bit was correct}) && (22)\\ q_1 &\overset{\text{def}}{=} \mathbb{P}(\text{stay with the same value}|\text{bit was incorrect}) && (23) \end{aligned}$$

Lemma 6. *For $b \in \{0,1\}$, we have*

$$q_b = O\left(\frac{(1-\varepsilon^2)^{w/4}}{\sqrt{\pi w}\varepsilon}\right).$$

Proof. For $b \in \{0,1\}$, we let X_1^b, $X_2^b, \ldots, X_{w/2}^b$ be independent random variables of parameter p_b. We obviously have

$$\begin{aligned} q_0 &\leq \mathbb{P}(\sum_{i=1}^{w/2} X_i^0 \geq w/4)\\ q_1 &\leq \mathbb{P}(\sum_{i=1}^{w/2} X_i^1 \leq w/4). \end{aligned}$$

By using Lemma 3 we obtain for q_0

$$\begin{aligned} q_0 &\leq \frac{(1-p_0)\sqrt{\frac{1}{2}}}{(\frac{1}{2}-p_0)\sqrt{2\pi\frac{w}{2}(1-\frac{1}{2})}} e^{-w/2D\left(\frac{1}{2}\|p_0\right)}\\ &\leq \frac{(1-p_0)}{\sqrt{\pi w}\varepsilon} e^{-w/2D\left(\frac{1}{2}\|\frac{1}{2}-\frac{1}{2}\varepsilon(1+O(1/w))\right)} && (24)\\ &\leq \frac{(1-p_0)}{\sqrt{\pi w}\varepsilon} e^{\frac{w\left(\ln(1-\varepsilon^2)+O\left(\frac{1}{w}\right)\right)}{4}} && (25)\\ &\leq O\left(\frac{(1-\varepsilon^2)^{w/4}}{\sqrt{\pi w}\varepsilon}\right) && (26) \end{aligned}$$

Whereas for q_1 we also obtain

$$\begin{aligned} q_1 &\leq \frac{p_1\sqrt{\frac{1}{2}}}{(p_1-\frac{1}{2})\sqrt{2\pi\frac{w}{2}\frac{1}{2}}} e^{-w/2D\left(\frac{1}{2}\|p_1\right)} && (27)\\ &\leq O\left(\frac{(1-\varepsilon^2)^{w/4}}{\sqrt{\pi w}\varepsilon}\right) && (28) \end{aligned}$$

□

A.4 Proof of Theorem 1

We are ready now to prove Theorem 1. We use here the notation of Assumption 1. Recall that e^0 denotes the true error vector. e^1 is the value of vector e after one round of iterative decoding in Algorithm 1. We let $\Delta e \overset{\text{def}}{=} e^0 + e^1$. Call $X_1^0, \dots, X_{n-t}^0$ the values after one round of iterative decoding of the $n-t$ bits which were without error initially (that is the bits i such that $e_i^0 = 0$) . Similarly let $X_1^1, \dots, X_t^1$ be the values after one round of iterative decoding of the t bits which were initially in error (i.e. for which $e_i^0 = 1$). We let

$$\begin{aligned} S_0 &\overset{\text{def}}{=} X_1^0 + \cdots + X_{n-t}^0 \\ S_1 &\overset{\text{def}}{=} X_1^1 + \cdots + X_t^1 \end{aligned}$$

S_0 is the number of errors that were introduced after one round of iterative decoding coming from flipping the $n-t$ bits that were initially correct, that is the number of i's for which $e_i^0 = 0$ and $e_i^1 = 1$. Similarly S_1 is the number of errors that are left after one round of iterative decoding coming from not flipping the t bits that were initially incorrect, that is the number of i's for which $e_i^0 = 1$ and $e_i^1 = 0$.

Let S be the weight of Δe. By assumption 1 we have

$$P_{\text{err}} \leq \mathbb{P}(|\Delta e| \geq \alpha t) = \mathbb{P}(S \geq \alpha t),$$

for some α in $(0, 1)$. We have

$$\begin{aligned} \mathbb{P}(S \geq \alpha t) &\leq \mathbb{P}(S_0 \geq \alpha t/2 \cup S_1 \geq \alpha t/2) \\ &\leq \mathbb{P}(S_0 \geq \alpha t/2) + \mathbb{P}(S_1 \geq \alpha t/2) \end{aligned}$$

By Assumption 1, S_0 is the sum of $n-t$ Bernoulli variables of parameter q_0. By applying Lemma 3 we obtain

$$\begin{aligned} \mathbb{P}(S_0 \geq \alpha t/2) &\leq \frac{(1-q_0)\sqrt{\frac{\alpha t}{2(n-t)}}}{(\frac{\alpha t}{2(n-t)} - q_0)\sqrt{2\pi(n-t)(1-\frac{\alpha t}{2(n-t)})}} e^{-(n-t)D\left(\frac{\alpha t}{2(n-t)} \middle\| q_0\right)} \\ &\leq \frac{1}{\sqrt{\alpha \pi t}} e^{-(n-t)D\left(\frac{\alpha t}{2(n-t)} \middle\| q_0\right)} \qquad (29) \end{aligned}$$

We observe now that

$$D\left(\frac{\alpha t}{2(n-t)} \middle\| q_0\right) \geq D\left(\frac{\alpha t}{2(n-t)} \middle\| O\left(\frac{(1-\varepsilon^2)^{w/4}}{\sqrt{\pi w}\varepsilon}\right)\right) \qquad (30)$$

where we used the upper-bound on q_0 coming from Lemma 6 and the fact that $D(x\|y) \geq D(x\|y')$ for $0 < y < y' < x < 1$. By using this and Lemma 2, we deduce

$$\begin{aligned} D\Big(\frac{\alpha t}{2(n-t)}\Big\|q_0\Big) &\geq \frac{\alpha t}{2(n-t)}\ln\left(\frac{\alpha t}{2(n-t)}\right) - \frac{\alpha t}{2(n-t)}\ln\left(O\left(\frac{(1-\varepsilon^2)^{w/4}}{\varepsilon\sqrt{w}}\right)\right) + O\left(\frac{\alpha t}{2(n-t)}\right) \\ &\geq \frac{\alpha t}{2(n-t)}\ln\left(\frac{t\sqrt{w}}{n}\right) - \frac{\alpha t w}{8(n-t)}\ln\left(1-\varepsilon^2\right) + O\left(\frac{t}{n}\right) \\ &\geq -\frac{\alpha t}{8(n-t)}\ln n - \frac{\alpha t w}{8(n-t)}\ln\left(1-\varepsilon^2\right) + O\left(\frac{t}{n}\right). \end{aligned}$$

By plugging in this expression in (29) we obtain

$$\mathbb{P}(S_0 \geq \alpha t/2) \leq \frac{1}{\sqrt{\alpha\pi t}} e^{\frac{\alpha t w}{8}\ln\left(1-\varepsilon^2\right) + \frac{\alpha t}{8}\ln(n) + O(t)}$$

On the other hand we have

$$\begin{aligned} \mathbb{P}(S_1 \geq \alpha t/2) &\leq \frac{(1-q_1)\sqrt{\frac{\alpha}{2}}}{(\frac{\alpha}{2}-q_1)\sqrt{2\pi t(1-\frac{\alpha}{2})}} e^{-tD\left(\frac{\alpha}{2}\|q_1\right)} \\ &\leq \frac{1}{\sqrt{\alpha\pi t}} e^{-tD\left(\frac{\alpha}{2}\|q_1\right)} \end{aligned} \tag{31}$$

Similarly to what we did above, by using the upper-bound on q_1 of Lemma 6 and $D(x\|y) \geq D(x\|y')$ for $0 < y < y' < x < 1$, we deduce that

$$D\Big(\frac{\alpha}{2}\Big\|q_1\Big) \geq D\left(\frac{\alpha}{2}\middle\|O\left(\frac{(1-\varepsilon^2)^{w/4}}{\varepsilon\sqrt{w}}\right)\right)$$

By using together with Lemma 2 we obtain

$$\begin{aligned} D\Big(\frac{\alpha}{2}\Big\|q_1\Big) &\geq -h(\alpha/2) - \frac{\alpha}{2}\ln\left(O\left(\frac{(1-\varepsilon^2)^{w/4}}{\varepsilon\sqrt{w}}\right)\right) + O\left(\frac{(1-4\varepsilon^2)^{w/4}}{\varepsilon\sqrt{w}}\right) \\ &\geq -\frac{\alpha w}{8}\ln\left(1-\varepsilon^2\right) + \frac{\alpha}{8}\ln n + O(1). \end{aligned}$$

By using this lower-bound in (31), we deduce

$$\mathbb{P}(S_1 \geq \alpha t/2) \leq \frac{1}{\sqrt{\alpha\pi t}} e^{\frac{\alpha t w}{8}\ln\left(1-\varepsilon^2\right) + \frac{\alpha t}{8}\ln(n) + O(t)}.$$

B Proof of Proposition 1

Let us recall first the 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.

Proof. Following Zalka[43], the best way is to perform Grover's algorithm sequentially with the maximum allowed number of iterations in order not to go beyond $\mathsf{MAXDEPTH}$. 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:

$$Q = \frac{1}{\alpha I^2} \cdot I \cdot C_f = \frac{D_f \cdot C_f}{\alpha \mathsf{MAXDEPTH}}. \tag{32}$$

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

$$Q_{AES-N} = \frac{2^N D_{AES-N} \cdot C_{AES-N}}{\mathsf{MAXDEPTH}}. \tag{33}$$

The proposition follows by comparing (32) with (33). □