BIKE:
Bit Flipping Key Encapsulation

Submission for Round 3 Consideration

Nicolas Aragon, University of Limoges, France

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

Slim Bettaieb, Worldline, France

Loïc Bidoux, Worldline, France

Olivier Blazy, University of Limoges, France

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

Philippe Gaborit, University of Limoges, France

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: 4.0

Release Date: May 3, 2020

Contents

1 Introduction

This document describes the Key Encapsulation Mechanism (KEM) based on Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) codes, that can be decoded using bit flipping decoding techniques, called **BIKE**.

This version presents an updated and simplified reference document. It is aimed at Round 3 of the Post-Quantum Cryptography Standardization project that is managed by the National Institute of Standards and Technology (NIST). Previous versions of this document are therefore to be regarded as a historical and technical repository. They remain available for the interested reader at the website: `https://bikesuite.org`.

1.1 Notation

Table 1 presents some notation used throughout the document.

Notation	Description
$\mathbb{F}_2$:	Finite field of 2 elements.
$\mathcal{R}$:	Cyclic polynomial ring $\mathbb{F}_2[X]/\langle X^r - 1\rangle$.
$\lvert v\rvert$:	Hamming weight of a binary polynomial v.
$\{0,1\}^l_{[t]}$	Set of all l-bit strings with Hamming weight t.
$u \xleftarrow{\$} U$:	Variable u is sampled uniformly at random from the set U.
h_j:	j-th column of a matrix H, as a row vector.
$\star$:	Component-wise product of vectors.
φ	Ring isomorphism between $(r \times r)$ circulant matrices and $\mathcal{R}$.

Table 1: Notation.

1.2 Preliminaries

This document uses the following definitions.

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

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

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

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

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

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

The mapping φ can be extended to any binary vector of $\mathbb{F}_2^r$. For all $\mathbf{v}=(v_0,v_1,\ldots,v_{r-1})$, we set $\varphi(\mathbf{v})=v_0+v_1X+\cdots+v_{r-1}X^{r-1}$. To stay consistent with the transposition, the image of the column vector $\mathbf{v}^T$ must be $\varphi(\mathbf{v}^T)=\varphi(\mathbf{v})^T=v_0+v_{r-1}X+\cdots+v_1X^{r-1}$. It is easy to see that $\varphi(\mathbf{v}A)=\varphi(\mathbf{v})\varphi(A)$ and $\varphi(A\mathbf{v}^T)=\varphi(A)\varphi(\mathbf{v})^T$.

The generator matrix of an (n_0,k_0)-QC code can be represented as a $k_0\times n_0$ 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 $O(\sqrt{n})$. Such a matrix allows the use of iterative decoders similar to those used for LDPC (Low-Density Parity-Check) codes [10], widely deployed for error correction in telecommunication. QC-MDPC codes are formally defined as follows.

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

We will discuss in Section 5 the techniques to efficiently correct errors for BIKE.

2 Algorithm Specification (2.B.1)

BIKE algorithms (Setup, KeyGen, Encaps, and Decaps), building blocks (decoder, random oracles, pseudorandom bits generation), and recommended parameters are defined here.

Setup

- Input: λ, the target quantum security level.
- Output: the set of parameters $\{r, w, t, \ell\}$ and hash functions $\{\mathbf{H}, \mathbf{K}, \mathbf{L}\}$.

1. Select r, w, t, ℓ in the following way.

   (a) r (block length): a prime s.t. $(X^r - 1)/(X - 1) \in \mathbb{F}_2[X]$ is irreducible.

   (b) w (row weight): an even positive integer such that $w/2$ is odd.

   (c) t (decoding radius): a positive integer.

   (d) ℓ (shared secret size): a positive integer.

   While not technically part of the scheme parameters, the following quantities are inherently defined by the Setup algorithm: n (code length), set to $n = 2r$ and d (block weight), set to $d = w/2$.

2. Select the functions $\mathbf{H}, \mathbf{K}, \mathbf{L}$ uniformly at random from the set of functions with the following respective domains and ranges.

   (a) $\mathbf{H} : \{0,1\}^{\ell} \to \{0,1\}^{2r}_{[t]}$.

   (b) $\mathbf{K} : \{0,1\}^{r+2\ell} \to \{0,1\}^{\ell}$.

   (c) $\mathbf{L} : \{0,1\}^{2r} \to \{0,1\}^{\ell}$

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

KeyGen

- Input: parameters $\{n, w, t, \ell\}$.
- Output: the private key (h_0, h_1, σ) and the public key h.

1. Generate $(h_0, h_1) \xleftarrow{\$} \mathcal{R}^2$ both of odd weight $|h_0| = |h_1| = w/2$.
2. Generate $\sigma \xleftarrow{\$} \{0,1\}^\ell$ uniformly at random.
3. Compute $h \leftarrow h_1 h_0^{-1}$.
4. Return (h_0, h_1, σ) and h.

Encaps

- Input: the public key h.
- Output: the encapsulated key K and the ciphertext $C = (c_0, c_1)$.

1. Generate $m \xleftarrow{\$} \{0,1\}^\ell$ uniformly at random.
2. Compute $(e_0, e_1) \leftarrow \mathbf{H}(m)$.
3. Compute $C = (c_0, c_1) \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$.
4. Compute $K \leftarrow \mathbf{K}(m, C)$.
5. Return (C, K).

Decaps

- Input: the private key (h_0, h_1, σ) and the ciphertext $C = (c_0, c_1)$.
- Output: the decapsulated key K.

1. Generate $(e'_0, e'_1) \xleftarrow{\$} \mathcal{R}^2$.
2. Compute the syndrome $s \leftarrow c_0 h_0$.
3. Compute† $\{(e''_0, e''_1), \perp\} \leftarrow \texttt{decoder}(s, h_0, h_1)$.
4. If $(e''_0, e''_1) \leftarrow \texttt{decoder}(s, h_0, h_1)$ and $|(e''_0, e''_1)| = t$ set $(e'_0, e'_1) \leftarrow (e''_0, e''_1)$.
5. $m' \leftarrow c_1 \oplus \mathbf{L}(e'_0, e'_1)$
6. If $\mathbf{H}(m') \neq (e'_0, e'_1)$ compute $K \leftarrow \mathbf{K}(\sigma, C)$.
7. Else compute $K \leftarrow \mathbf{K}(m', C)$.
8. Return K.

† Invoke a decoding algorithm `decoder` that is associated with the scheme (see next section). It returns either an error vector or a failure symbol $\perp$.

2.1 Decoding Failure Rate

Let `decoder` be a decoding algorithm used by Decaps. The algorithm takes some values s, h_0, h_1 as input and returns an error vector $(e_0, e_1) \in \mathcal{R}^2$ or a decoding failure indication, via the special failure symbol $\perp$. Such a decoding failure occurs when the recovered error vector does not match the input syndrome, i.e. $e_0h_0 + e_1h_1 = s$, or it doesn't have the correct weight t. Accordingly, the Decoding Failure Rate of `decoder` is the probability that `decoder`(s, h_0, h_1) does not produce (e_0, e_1) on input $(s = e_0h_0 + e_1h_1,\ h_0,\ h_1)$, when e_0, e_1, h_0, h_1 are chosen uniformly such that $|h_0| = |h_1| = w/2$ and $|e_0| + |e_1| = t$.
For short, the Decoding Failure Rate is also called the DFR of `decoder`. In the context of a specific decoder, it is further shortened to simply the DFR. The DFR is a property of the decoding algorithm (not of the KEM).

2.2 The Black-Gray-Flip (BGF) Decoder

The decoding algorithm (aka decoder) that is associated with the instantiation of BIKE is the Black-Gray-Flip (BGF) defined in [8]. It is described in Algorithm 1 using, for convenience, matrix notation (as in Definition 2 and 3).
The algorithm is defined for every set of system parameters (r, w, t) (that also determine $d = w/2$ and $n = 2r$). It is characterized by three other parameters. The first is NbIter - the number of iterations that it runs, and the second is τ - a threshold gap. The third parameter is the threshold function (see below). Relevant values of NbIter, τ, and of the threshold function must be specified for every parameter set.
The algorithm invokes two functions.

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

- **threshold**(S, i). This function is the threshold selection rule. It depends, in general, on the syndrome weight S, the iteration number i, and on the system parameters. This function may vary and is a parameter of the algorithm. Its current value (given below) is independent of r and i.

The parameters used here with the BGF decoder are:

- For Level 1: NbIter $= 5$, $\tau = 3$,
  threshold$(S, i) = \max(\lceil 0.0069722 \cdot S + 13.530 \rceil, 36)$

- For Level 3: NbIter $= 5$, $\tau = 3$,
  threshold$(S, i) = \max(\lceil 0.005265 \cdot S + 15.2588 \rceil, 52)$

Details on the design rationale, analysis, parameters choice, threshold function, and DFR estimation, are provided in Section 5.

Algorithm 1 Black-Gray-Flip (BGF)

```
Parameters: $r$, $w$, $t$, $d = w/2$, $n = 2r$ ; NbIter, $\tau$, threshold (see text for details)
Require: $H \in \mathbb{F}_2^{r \times n}$, $s \in \mathbb{F}_2^r$
1: $e \leftarrow 0^n$
2: for $i = 1, \ldots, \text{NbIter}$ do
3:     $T \leftarrow$ threshold$(\left|s + eH^T\right|, i)$
4:     $e, \text{black}, \text{gray} \leftarrow$ BFIter$(s + eH^T, e, T, H)$
5:     if $i = 1$ then
6:         $e \leftarrow$ BFMaskedIter$(s + eH^T, e, \text{black}, (d+1)/2 + 1, H)$
7:         $e \leftarrow$ BFMaskedIter$(s + eH^T, e, \text{gray}, (d+1)/2 + 1, H)$
8: if $s = eH^T$ then
9:     return $e$
10: else
11:     return $\perp$

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

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

2.3 Pseudorandom Bits Generation

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

- With no constraints on the output (Alg. 2).
- With odd weight (Alg. 3).
- With a specific weight w (Alg. 4).

Remark 1. *Alg. 4 is a "Rejection Sampling" method. It generates a list of w distinct positions between 0 and $r-1$. This list is also viewed, interchangeably, as the support of a string U of r bits (where $|U| = w$).*

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

Algorithm 2 GenPseudoRand(seed, len)

```
Require: seed (32 bytes)
1: return AES-CTR-Stream (seed, len)
```

Algorithm 3 GenPseudoRandOddWeight(seed, len)

```
Require: seed (32 bytes), len
1: z = GenPseudoRand(seed, len)
2: if $|z|$ is even then z[0] = z[0] $\oplus$1    ($z[0]$ is the least significant bit of $z$)
3: return $z$
```

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

```
Require: wt (32 bits), len
Ensure: A list (wlist) of wt bit-positions in $[0, \dots, \mathsf{len} - 1]$.
1: wlist$= \phi$; $ctr = 0$; $i = 0$
2: $s =$ AES-CTR-Stream(seed, $\infty$)          ▷ $\infty$ denotes "sufficiently large"
3: $mask = (2^{\mathsf{ceil}(log_2 r)} - 1)$
4: while $ctr <$ wt do
5:     $pos = s[32(i+1) - 1 : 32i]$ & $mask$          ▷ & denotes bitwise AND
6:     if ((pos $<$ len) AND (pos $\notin$ wlist)) then
7:         wlist $=$ wlist $\cup$ {pos}; $ctr = ctr + 1$; $i = i + 1$
8: return wlist, s
```

2.4 The Functions $\mathbf{H}, \mathbf{K}, \mathbf{L}$

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

- **H** is instantiated as a pseudorandom expansion of a seed of length ℓ bits that is input to the function. It is generated by invoking Alg. 4 with the approproate parameters.

- **K** is instantiated as the $\ell = 256$ least significant bits of the standard SHA384 hash digest of the input. The notation $\mathbf{K}(m, C)$ where $C = (c_0, c_1)$ (and similarly, $\mathbf{K}(m', C)$) refers to hashing an input of $\{0,1\}^{\ell+r+\ell}$ bits that is the concatenation of m, c_0 and c_1. Here, the bits of m are consumed (by SHA384) first, then the bits of c_0, and then the bits of c_1.

- **L** is instantiated as the $\ell = 256$ least significant bits of the standard SHA384 hash digest of the input. The notation $\mathbf{L}(e_0, e_1)$ (and similarly, $\mathbf{L}(e'_0, e'_1)$) refers to hashing an input of $\{0,1\}^{r+r}$ bits that is the concatenation of e_0 and e_1 Here, the bits of e_0 are consumed (by SHA384) first, and then then the bits of e_1.

2.5 BIKE Parameters

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

For all security levels, the key length parameter is fixed to $\ell = 256$. A parameter set for BIKE is a triple (r, w, t). The suggested parameters are summarized in Table 2.

Security	r	w	t	DFR
Level 1	12,323	142	134	2^{-128}
Level 3	24,659	206	199	2^{-192}

Table 2: Suggested BIKE Parameters. The Decoding Failure Rate (DFR) is a characteristic of the specific decoder used in a given instantiation of BIKE, as explained in Section 2.1, above. The DFR values in the table indicate a target DFR that a decoder needs to meet by the proposed design.

3 The Security of BIKE (2.B.4)

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

BIKE with the BGF decoder

BIKE instantiated with the BGF decoder is an IND-CPA secure KEM.

A formal argument is given in Appendix A.

BIKE with a decoder that has a provably low DFR

Suppose that BIKE is instantiated with a decoding algorithm `decode` where:

1. `decode` has a DFR of 2^{-128} for Level-1 (and 2^{-192} for Level-3).
2. `decode` runs in a constant number of steps.
3. The probability for a decoding success but a decryption failure is negligible.

Then, the resulting BIKE instantiation is an IND-CCA secure KEM.

A proof is given in [9] and a summary is included in Appendix A.

Practical security considerations for using BIKE

- BIKE is designed to be used with ephemeral keys. The party that initiates a session needs to: a) Generate a fresh private/public key pair for every session; b) Refuse to decapsulate more than one incoming ciphertext (presumably the result of a legitimate encapsulation) with that key. The IND-CPA security property suffices for this type of usage.
- An instantiation of BIKE can use different decoders without affecting interoperability. If a (practical) decoder with a proven upper bound for a sufficiently low DFR that can be implemented in constant-time is found and used, the instantiation of BIKE with this decoder would be IND-CCA secure.
- The DFR of the BGF decoder has been studied by means of simulations and extrapolations, and the details are provided in Section 5. These techniques provide a strong indication that the DFR is (sufficiently) small with the recommended parameters. This indication may be acceptable from a practical viewpoint, and could be strengthened by further studies. However, at the moment, the current analysis gives only an *estimation* of the DFR, and not a proven upper bound. Consequently, the BIKE instantiation with the BGF decoder does not make a formal claim for IND-CCA security, although by any practical considerations, this is probably the case.
- An IND-CCA secure KEM could support a long-term use of a single key. However, this usage model implies the loss of forward secrecy.

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

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

4.1 What is BIKE and how should it be used?

4.1.1 What is BIKE in one sentences?

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

4.1.2 How many versions of BIKE are proposed?

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

4.1.3 What is the security claim of BIKE?

When BIKE is instantiated with the BGF decoder and the recommended parameters, it is an IND-CPA secure KEM. It targets Security Levels 1 and 3, as defined in the NIST call for proposals. There is some probability that a session using BIKE would fail, i.e., not end up with a successfully agreed shared key. BIKE design target is to make this probability at most 2^{-128} for Security Level 1 and 2^{-192} for Security Level 3.

4.1.4 How should BIKE be used?

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

4.1.5 Is BIKE an IND-CCA secure KEM?

If BIKE is used with a decoder that has a Decoding Failure Rate (DFR) of the required magnitude, then it is an IND-CCA secure KEM [9]. The BGF decoder that is associated with BIKE targets a DFR of 2^{-128} and 2^{-192} for the respective levels of security. This is indeed the *estimated* DFR. The estimation suggests a high confidence level through a methodology that uses extensive simulations and extrapolations. Since currently there is no formal proof for an upped bound on the DFR, BIKE is not declared formally as an IND-CCA secure KEM.

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

This scenario is a violation of the recommended use of BIKE. Formally, unless there is a proof for the DFR of the decoder, this affects the security guarantee. Nevertheless, it does not seem unreasonable to believe that such a violation would not open the door to a practical exploitation.

4.2 Interoperability

4.2.1 Can BIKE be used with a different decoder?

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

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

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

4.2.3 Can BIKE be used with another pseudorandom generator?

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

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

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

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

4.3 Design rationale

4.3.1 How is BIKE constructed?

BIKE is built upon the Niederreiter framework, with some tweaks. It also applies the implicit-rejection version of Fujisaki-Okamoto transformation ($FO^{\not\perp}$, as described in [13]) for converting a δ-correct PKE into an IND-CCA KEM.

4.3.2 What happened to the previous versions of BIKE?

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

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

Not exactly, because some minor changes were introduced in the Encaps and Decaps algorithms. These changes are recommended in [9], in order to prove that BIKE is IND-CCA secure if the associated decoder has a sufficiently low DFR. Details are provided in the Formal Security Discussion (Appendix A) .

4.3.4 Why keep the Fujisaki-Okamoto transformation?

This is a design choice that targets simplicity. Indeed, it is possible to build a version of BIKE that does not apply the $FO^{\not\perp}$ transformation and targets only IND-CPA security. However, the difference in the performance is negligible (see [6]) and does not justify the complication of maintaining such a design as a separate version.

4.3.5 Why is BIKE designed over the Niederreiter framework?

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

4.3.6 How can BIKE support polynomial inversion in KeyGen?

The cost of polynomial inversion was considered too prohibitive until recently (especially with ephemeral keys usage), but the fast polynomial inversion algorithm proposed in [7] changed the picture. This algorithm is similar to the Itoh-Tsuji inversion algorithm, where the essence is that computing a^{2^k} is efficient. The Itoh-Tsuji algorithm inverts an element of $\mathbb{F}_{2^k}$, where the field elements are represented in normal basis. The new algorithm generalizes it to the ring of polynomials used in BIKE (and other QC-MDPC schemes): $\mathbb{F}_2[x]/\langle (x-1)h \rangle$ with irreducible h. Details are provided in [7]. This algorithm is implemented in constant-time and used in the Additional Software Implementation Code Package (see Section 7.3).

4.3.7 How was the block length r chosen?

The block length r determines the sizes of the public key, the ciphertext, and significantly affects the overall latency and the communication bandwidth. By the design of BIKE, r needs to be prime and satisfy the requirement that $(X^r-1)/(X-1) \in \mathbb{F}_2[X]$ is irreducible. It needs to be sufficiently large to satisfy (together with the choice of w and t) the scheme's security target and the DFR target for the decoder. In addition, [7] suggests that the inversion algorithm is especially efficient if the Hamming weight of $(r-2)$, is small. Indeed, for $r = 12323$, $|(r-2)| = 4$, and for $r = 24659$ $|(r-2)| = 5$.

4.3.8 How was the pseudorandom generation determined?

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

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

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

5 Decoding algorithms and DFR estimation

5.1 Preliminaries

The decoding algorithm (decoder) is a critical element of the decapsulation algorithm (Decaps) of BIKE. Its purpose is to find the unique solution of a decoding problem. During a key exchange session, the initiating party executes KeyGen and sends the public key to the responding party that is expected to send back some ciphertext. Subsequently, the initiating party executes Decaps, which, along with other steps specified in Section 2, invokes the decoding algorithm.

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

Decoder Description: For BIKE, we will consider the decoding as a black box running in bounded time and which either returns a valid error pattern or fails. As we describe it, it takes as arguments a (sparse) parity check matrix $H \in \mathbb{F}_2^{r \times n}$ and a syndrome $s \in \mathbb{F}_2^r$. When the decoder does not fail, the returned value e is such that $s = eH^T$.

For given BIKE parameters r, w, t, any $h \in \mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$ will be identified with the $r \times r$ circulant matrix whose first row is h. The decoder input (H, s) is such that:

- the matrix H is block-circulant of index 2, that is $H = (h_0^T \quad h_1^T)$ with $(h_0, h_1) \in \mathcal{R}^2$ such that $|h_0| = |h_1| = w/2$
- the syndrome s is equal to $e_0h_0 + e_1h_1$ for some $(e_0, e_1) \in \mathcal{R}^2$ such that $|e_0| + |e_1| = t$.

The decoder defined in Section 2.1 fails if it does not return (e_0, e_1) on input $(s = e_0h_0 + e_1h_1, h_0, h_1)$. The Decoding Failure Rate is defined as the probability for the decoder to fail when the input is $(s = e_0h_0 + e_1h_1, h_0, h_1)$ with (h_0, h_1, e_0, e_1) distributed uniformly such that $|h_0| = |h_1| = w/2$ and $|e_0| + |e_1| = t$.

5.2 Black-Gray Decoding

The authors of [6] discussed the importance of defining a decoder as an algorithm that runs a fixed number of steps (rather then a maximal number of steps). Such a definition also makes the algorithm implementable in constant-time, which is a required property from a cryptographic primitive. Of course, a real application needs to actually use a concrete constant-time implementation. In addition, [6] also identified the Black-Gray decoder as providing a favorable trade-off between: a) the number of steps; b) the estimated resulting DFR; c) the performance of a constant-time implementation. The subsequent publication [8] by the same authors defined several variants of the Black-Gray decoder, and studied the resulting trade-offs. One variant is the Black-Gray-Flip (BGF) decoder that starts with one Black-Gray iteration and continues with several Bit-Flipping iterations. It was identified in [8] as the most efficient variant, at least for the studied platforms (see Algorithm 1 in

[8]). BIKE uses the BGF decoder with tuned threshold functions that are based on fresh extensive simulations.

Threshold Selection Rule `threshold`(S, i)**.** The rule that is currently used derives from BIKE Round 1. In practice, for each security level it is given as an affine function of the syndrome weight. The numerical values are given in Section 2.2 for Level 1 and 3. The coefficients of the current affine functions depend on the system parameters w and t, but not on r. The current rule do not depend of the iteration number i either. Other strategies, depending on i and r are possible. Experiments indicate that those more elaborated strategies do not perform better (for the BGF decoder). Our simulations and estimated DFR claims are based on the rules given in the specification.

5.3 Estimating the DFR for High Block Size

The Low Impact of Block Size on Computational Assumptions. The block size r must be chosen large enough to allow efficient decoding. In practice one must choose $r = \Omega(wt)$. The higher r the lower the DFR. On the other hand, the best known attacks for codes of rate 1/2 as we have here, are of order $2^{t(1+o(1))}$ or $2^{w(1+o(1))}$. This is corrected by a factor polynomial in r which is very small in practice. An interesting consequence is that if w and t are fixed, a moderate modification of r (say plus or minus 50%) will not significantly affect the resistance against the best known key and message attacks. This will simplify the extrapolation methodology described in the next paragraph.

Estimating the DFR by Extrapolation. Low DFR, *e.g.,* 2^{-128}, as required for CCA security, cannot be directly estimated by simulation. Instead, simulations are combined with extrapolations, as described next. First, the DFR is measured for smaller block sizes r, for which simulations are meaningful (and assumed to provide a reliable estimation). Subsequently, one can define a curve based on the sample of $r - DFR$ acquired values, an the curve is extrapolated to a larger block size for which the extrapolated DFR reaches the target. Known asymptotic models for simpler variants of bit flipping, as [21, 19], predict a concave shape for the curve in the relevant range of r values. Assuming a similar behavior, as described in [20], a linear extrapolation over two (acquired) points shoots to an overestimation of the required r (i.e., a conservative estimation). More extensive simulations can refine the DFR estimation and hence lead to smaller (more desirable) sufficient r. References [6] and [8] discuss simulation results with different extrapolations for several decoders, including the Black-Gray variants that are used for BIKE.

6 Known Attacks (2.B.5)

This section discusses the practical security aspects of our proposal.

6.1 Hard Problems and Security Reduction

In the generic (*i.e.* non quasi-cyclic) case, the two following problems were proven NP-complete in [3].

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.

6.1.1 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, which are of interest to us, we do not know whether or not SD and CF are NP-complete.

Nevertheless, the problems 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, and 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 codes, the parity check matrix (over $\mathcal{R}$) has the form

$$\begin{pmatrix} 1 & h \end{pmatrix} \text{ with } h \in \mathcal{R}.$$

For BIKE, we are interested only in the above type of QC codes and to the two 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 $c_0 + c_1 h = 0$.*

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. As they are presented, the problems have the appearance of *sparse polynomials* problem, but in fact they are equivalent to the generic quasi-cyclic decoding and codeword finding problems.

For our security proof, we will use the decisional versions of the $(2,1)$-QCSD and $(2,1)$-QCCF problems, instead of their search versions given in Problems 3 and 4, respectively. We argue that the search and decisional versions of these problems have similar hardness.

The message security for BIKE relies 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 $e_0 + e_1 h = c$.*

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 [3]. 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.

The key indistinguishability for BIKE requires a balanced variant of Problem 4, as defined next.

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 $h_0 + h_1 h = 0$.*

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 1 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.*

In the current state of the art, the best known techniques for solving those problems are variants of Prange's Information Set Decoding (ISD) [17]. 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.

6.2 Attacks

6.2.1 Information Set Decoding

The best asymptotic variant of ISD is due to May and Ozerov [16], but it has a polynomial overhead which is difficult to estimate precisely. In practice, the BJMM variant [2] 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 [22] 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 [12] 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.

6.2.2 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) [18] 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.

6.2.3 Exploiting Quantum Computations.

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

1. A quantum computer can only perform quantum computations of limited depth. They introduce a parameter, MAXDEPTH, which can range from 2^{40} to 2^{96}. This accounts for the practical difficulty of building a full quantum computer.

2. The amount (or bits) of security is not measured in terms of absolute time but in the time required to perform a specific task.

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

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

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

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

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

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

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 [4]). Theoretically there is a slightly better algorithm consisting in quantizing more sophisticated ISD algorithms [14], 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.

6.2.4 The GJS Reaction Attack

BIKE is currently designed to use ephemeral keys. This usage defeats the GJS *reaction attack* [11]. Indeed, an adversary has (at most) a single opportunity to submit a decryption query, and this does not allow to create statistics on different error patterns for a specific key.

6.3 Choice of Parameters

Let $\mathrm{WF}(n, k, t)$ denote the workfactor of the best ISD variant for decoding t errors in a binary code of length n and dimension k. Hereafter, only codes of transmission rate 0.5 (i.e., length $n = 2r$ and dimension r) are considered. 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 $\mathrm{WF}(2r, r, w)/r$.

As remarked above, with $\mathrm{WF}(n,k,\ell) \approx 2^{\ell \log_2 \frac{n}{n-k}}$ gives a crude but surprisingly accurate, parameter selection rule. The target security levels correspond to AESλ with $\lambda \in \{128, 192\}$. To reach λ bits of classical security, w, t and r are chosen such that

- Problem 3 with block size r and weight t and Problem 4 with block size r and weight w must be hard enough. In other words,

$$\lambda \approx t - \frac{1}{2}\log_2 r \approx w - \log_2 r. \qquad (1)$$

This equation has to be solved in addition to the constraint that r must be large enough to decode t errors in $(2,1,r,w)$-QC-MDPC code, with a small failure rate. Finally, r is chosen such that: a) 2 is primitive modulo r; b) r is a prime number (note that 2 is primitive modulo r does not imply primality. A counter example is $r = 10201$ which is composite).

This choice thwarts the so-called squaring attack [15]. Also, it implies that $(X^r - 1)$ has only two irreducible factors (one of them is $X - 1$). 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 the discussed system. By §6.2.3, it follows that the proposed set of parameters correspond the Security Levels 1 and 3, as described in the NIST call for quantum safe primitives.

7 BIKE Performance (2.B.2)

This section discusses the essential characteristics and performance of BIKE.

7.1 Memory and Communication Bandwidth

Table 3 summarizes the minimum memory requirements for BIKE.

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

Table 3: Private Key, Public Key and Ciphertext sizes (in bits).

Remark 3. The private key consists of the vectors $(h_0, h_1) \in \mathcal{R}$ with $|h_0| = |h_1| = w/2$ and (σ). Both h_0 and h_1 can be represented by r bits. Alternatively, a more compact representation is listing the $w/2$ positions of the set bits. This listing yields a $(\frac{w}{2} \cdot \lceil \log_2(r) \rceil)$-bits representation. Therefore, the size for this part of the private key is $(w \cdot \lceil \log_2(r) \rceil)$-bits. Since $\lceil \log_2(r) \rceil < 16$ for the proposed parameter sets, an implementation may prefer (for simplicity) to store these vectors as a sequence of w 16-bits elements. The second part of the private key, (σ), requires ℓ bits of storage. In total, BIKE private keys can be stored in a container of $(\ell + w \cdot \lceil \log_2(r) \rceil)$ bits. Applications may choose to explore the possibility of generating the private key on the fly, from a (secured) seed to obtain a favorable memory vs. latency trade-off.

Table 4 shows the communication bandwidth cost per message.

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

Table 4: BIKE communication bandwidth (in bits).

7.2 Reference Implementation

The reference implementation of BIKE is available at `https://bikesuite.org/reference.html`. It is a pure C implementation intended to provide readability and help researchers get familiarized with the BIKE algorithms. It is not designed to run in constant-time, as required for real-world implementation to offer side-channel resistance. For real-world performance characterization, the reader is referred to the Additional Implementation numbers described in section 7.3, which is constant-time and leverages efficient platform instruction sets.

7.3 Additional Software Implementation

The Additional Software Implementation Code Package for BIKE was developed by Nir Drucker, Shay Gueron, and Dusan Kostic. It is maintained in the github repository `github.com/awslabs/bike-kem`.

The package includes the following implementations:

1. PORTABLE: a C (C99) portable code implementation.

2. AVX2: implementation that leverages the AVX2 architecture features. It is written in C (with C intrinsics for AVX2 functions).
3. AVX512: implementation that leverages the AVX512 architecture features. It is written in C (with C intrinsics for AVX512 functions). This implementation can also be compiled to use the latest `vector-PCLMULQDQ` instruction that is available on the Intel IceLake processors.

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

Performance benchmarking details. The performance is reported here in processor cycles, and reflects the performance per *single core*. The measurements methodology follows the description in [5].

The benchmarking platform. The platform used in the experiments was equipped with 10th generation Intel®Core^{TM} processor (microarchitecture codename "Ice Lake"[ICL]). The machine is Dell XPS 13 7390 2in1 laptop with Intel®Core^{TM} i7-1065G7 CPU working at 1.30GHz, with 16 GB RAM, 48K L1d cache, 32K L1i cache, 512K L2 cache, and 8MiB L3 cache. The CPU supports AVX512 instruction set and `vector-PCLMULQDQ` instruction. The Intel® Turbo Boost Technology was turned off for the experiments in order to force a fixed frequency and consistently measure performance in processor cycles.

OS and compilation. The code was compiled with gcc (version 9.2.1) and ran on a Linux OS (Ubuntu 19.04).

Performance numbers

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

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

Table 6: BIKE Level-3, $r = 24659$, $w = 206$, $t = 199$. Decoder BGF with 5 iterations. Performance in 10^3 cycles.

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

Remark 4. *A meaningful measure for the efficiency of the KEM, in the case where it is used with ephemeral keys is the cumulative latency of KeyGen and Decaps. The reason is that the communicating party that initiates the exchange executes KeyGen subsequently executes Decaps. The numbers reported in Tables 5 and 6 indicate that KeyGen is significantly faster than Decaps on modern platforms with AVX2 and AVX512 support. This property is due to the* `PCLMULQDQ` *instruction, and even more so to the newer* `vector-PCLMULQDQ` *instruction.*

7.4 Hardware Implementation

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

1. Reference implementation for Key Generation Level 1
2. Reference implementation for Key Generation Level 3
3. Reference implementation for Encapsulation will be added as soon as possible
4. Reference implementation for Decapsulation will be added as soon as possible

All the hardware files are published on the BIKE website and can be found here https://bikesuite.org/. The key generation design is challenging due to the costly polynomial inversion.

Implementation Results The implementation results are summarized in Table 7 including hardware utilization and timing behavior. All results were generated for an Artix-7 FPGA (xc7a100). Since the hardware implementation consumes only 590 slices and four BRAMs, it is perfectly suited for small devices.

Table 7: Implementation results on an Artix-7 (xc7a100) FPGA for the key generation.

	Resources				**Performance**		
	Logic	Memory		Area	Cycles	Frequency	Latency
	LUT	FF	BRAM	Slices	Cycles (average)	MHz	ms
Level 1	1 865	589	4	590	7 370 429	135	54.54
Level 3	1 884	557	5	593	30 447 947	131	231.4

Estimations The multiplier required for the key generation and encapsulation was highly optimized in order to provide a higher throughput. Based on this module, Table 8 provides performance numbers for the estimated latency (in clock cycles) for the hardware implementation of the encapsulation. The parameter d defines the internally used bus width and can be chosen from the set $\mathcal{D} \in \{32, 64, 128\}$. Theoretically larger values are possible but the available hardware resources would be exceeded.

The work regarding an implementation of the decapsulation is currently ongoing. Implementation results will be added as soon as possible.

Table 8: Estimated latency in clock cycles for the encapsulation for both security levels and for different d.

Security	$d = 32$	$d = 64$	$d = 128$
Level 1	162 000	50 000	22 000
Level 3	610 000	164 000	52 000

8 Known Answer Tests – KAT (2.B.3)

8.1 KAT for BIKE

The KAT files of BIKE are available in:

- req file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level1_3114.req`
- rsp file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level1_3114.rsp`
- req file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level3_6198.req`
- rsp file: `KAT/INDCPA/BIKE/PQCkemKAT_BIKE1-Level3_6198.rsp`

9 Acknowledgments

Special thanks are extended to Nir Drucker and Dusan Kostic for significant contributions that helped shaping BIKE into its current form. They co-authored the following papers [6] (accepted to CBCrypto 2020), [8] (accepted to PQCrypto 2020), [7] (accepted to CSCML 2020), [9]. They are also co-developers of the Additional Software Implementation Code Package of BIKE (residing in the git repository `github.com/awslabs/bike-kem`).

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

Paulo S. L. M. Barreto was also partially supported by Intel and FAPESP through the project "Efficient Post-Quantum Cryptography for Building Advanced Security Applications" (grant No. 2015/50520-6).

Shay Gueron was also partially supported by: NSF-BSF Grant 2018640; The Israel Science Foundation (grant No. 3380/19); The BIU Center for Research in Applied

Cryptography and Cyber Security, and the Center for Cyber Law and Policy at the University of Haifa, both in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

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

Edoardo Persichetti was partially supported by NSF Grant CNS 1906360.

References

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

[2] 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.

[3] 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.

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

[5] Nir Drucker and Shay Gueron. A toolbox for software optimization of QC-MDPC code-based cryptosystems. *Journal of Cryptographic Engineering*, pages 1–17, Jan. 2019.

[6] Nir Drucker, Shay Gueron, and Dusan Kostic. On constant-time QC-MDPC decoding with negligible failure rate. Cryptology ePrint Archive, Report 2019/1289, Nov 2019.

[7] Nir Drucker, Shay Gueron, and Dusan Kostic. Fast polynomial inversion for post quantum qc-mdpc cryptography. Cryptology ePrint Archive, Report 2020/298, 2020. https://eprint.iacr.org/2020/298.

[8] Nir Drucker, Shay Gueron, and Dusan Kostic. QC-MDPC decoders with several shades of gray. In Jintai Ding and Jean-Pierre Tillich, editors, *PQCrypto 2020*, volume 12100 of *LNCS*, pages 35–50. Springer, 2020.

[9] Nir Drucker, Shay Gueron, Dusan Kostic, and Edoardo Persichetti. On the applicability of the Fujisaki-Okamoto transformation to the BIKE KEM. *IACR Cryptology ePrint Archive*, 2020.

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

[11] 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.

[12] 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`.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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

[19] Nicolas Sendrier and Valentin Vasseur. On the decoding failure rate of QC-MDPC bit-flipping decoders. In Jintai Ding and Rainer Steinwandt, editors, *PQCrypto 2019*, volume 11505 of *LNCS*, pages 404–416, Chongquing, China, May 2019. Springer.

[20] Nicolas Sendrier and Valentin Vasseur. About low DFR for QC-MDPC decoding. In Jintai Ding and Jean-Pierre Tillich, editors, *PQCrypto 2020*, volume 12100 of *LNCS*, pages 20–34. Springer, 2020.

[21] Jean-Pierre Tillich. The decoding failure probability of MDPC codes. In *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*, pages 941–945, 2018.

[22] 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.

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

A Formal Security Discussion

The BIKE protocol flow is obtained by composing a "basic" KEM[1] with a one-time pad to define an IND-CPA secure PKE, which is then turned into the "final" KEM with some additional technical steps such as re-encryption and integrity checks. The initial KEM follows Niederreiter's framework with a systematic parity check matrix, and uses an MDPC decoder (such as that given in Algorithm 1). The parameters $\{r, w, t, \ell\}$ and the hash function $\mathbf{L}$ are obtained using the same Setup algorithm given at the beginning of Section 2. The KEM flow is described below.

KeyGen

- Input: parameters $\{r, w, t, \ell\}$.
- Output: the private key (h_0, h_1) and the public key h.

1. Generate $(h_0, h_1) \xleftarrow{\$} \mathcal{R}^2$ both of odd weight $|h_0| = |h_1| = w/2$.
2. Compute $h \leftarrow h_1 h_0^{-1}$.
3. Return (h_0, h_1) and h.

Encaps

- Input: the public key h.
- Output: the encapsulated key K and the ciphertext 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 $K \leftarrow \mathbf{L}(e_0, e_1)$.
4. Return (c, K).

Decaps

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

1. Compute the syndrome $s \leftarrow ch_0$.
2. Compute $\{(e'_0, e'_1), \perp\} \leftarrow \texttt{decoder}(s, h_0, h_1)$.
3. If $\perp \leftarrow \texttt{decoder}(s, h_0, h_1)$ or $|(e'_0, e'_1)| \neq t$, output $\perp$ and halt.
4. Else, compute $K \leftarrow \mathbf{L}(e'_0, e'_1)$ and return K.

[1]This was originally known as BIKE-2 in earlier versions of this document.

We now show that this KEM is IND-CPA secure, which is at the basis of the security for the final scheme. We start by recalling the definition of IND-CPA security for a KEM, where we denote by $\mathcal{K}$ the domain of the exchanged symmetric keys and by λ the security level.

Definition 7. *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 1. *The KEM described above is IND-CPA secure in the Random Oracle Model under the* $(2,1)$*-QCCF and* $(2,1)$*-QCSD assumptions.*

Proof. To begin, note that we model the hash function $\mathbf{L}$ 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 6.1 for definitions).

First let us instantiate the IND-CPA game for the underlying KEM. The game will use the following randomness

$$\begin{cases} (h_0, h_1) & \xleftarrow{\$} & \mathcal{R}^2 & |h_0| = |h_1| = w/2 \text{ odd} \\ (e_0, e_1) & \xleftarrow{\$} & \mathcal{R}^2 & |e_0| + |e_1| = t \end{cases}$$

The output (sk, pk) of $\textsc{Gen}(\lambda)$ will be $sk = (h_0, h_1)$ for all variants and $pk = h = h_1 h_0^{-1}$. For both valid and random pk, the output (c, K) of $\textsc{Encaps}(pk)$ will be $K = \mathbf{L}(e_0, e_1) \in \mathcal{K} = \{0,1\}^{\ell_K}$ and $c = e_0 + e_1 h$.

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

Game $\boldsymbol{G}_1$: 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 $\boldsymbol{G}_2$: 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 the honestly-generated public key in step 1 is replaced by $pk = h \overset{\$}{\leftarrow} \mathcal{R}$, for $|h|$ odd. 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 in fact distinguish $h_1 h_0^{-1}$ from a random invertible element of $\mathcal{R}$. Thus, we have that $\mathrm{Adv}^{\mathbf{G}_1}(\mathcal{A}) \leq \mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) + \mathrm{Adv}^{(2,1)-\mathrm{QCCF}}(\mathcal{A}')$ and therefore

$\mathrm{Adv}^{\mathbf{G}_1}(\mathcal{A}) - \mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) \leq \mathrm{Adv}^{(2,1)\text{-QCCF}}(\mathcal{A}')$ 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 the ciphertext in step 4 by $c^* \overset{\$}{\leftarrow} \mathcal{R}$, for $|c^*|$ odd. 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 in fact, given h random in $\mathcal{R}$ of odd weight, be able to distinguish $e_0 + e_1 h$ from a random element of $\mathcal{R}$ of identical parity. Thus, $\mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) \leq \mathrm{Adv}^{\mathbf{G}_3}(\mathcal{A}) + \mathrm{Adv}^{(2,1)-\mathrm{QCSD}}(\mathcal{A}'')$, where $\mathcal{A}''$ is a polynomial-time adversary for the underlying problem. It follows that, 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 $\mathrm{Adv}^{\mathbf{G}_2}(\mathcal{A}) - \mathrm{Adv}^{\mathbf{G}_3}(\mathcal{A}) \leq \mathrm{Adv}^{(2,1)\text{-QCSD}}(\mathcal{A}'')$

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 $\mathbf{L}$ is modeled as a random oracle[2], and therefore the adversary only has negligible advantage, say ϵ. So in the end, we have:

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

This concludes the proof.

□

[2]To nitpick, one could simply pick $\mathbf{L}$ as any Key Derivation Function, however for efficiency purposes it is simpler to consider it as a Random Oracle.

We now proceed to present the hybrid encryption scheme constructed using the KEM we just described. This is formally the PKE underlying the BIKE KEM, obtained through the Fujisaki-Okamoto transformation. Once again, the parameters $\{r, w, t, \ell\}$ and the hash function $\mathbf{L}$ are obtained using the same Setup algorithm given at the beginning of Section 2.

KeyGen

- Input: parameters $\{r, w, t, \ell\}$.
- Output: the sparse private key (h_0, h_1) and the dense public key h.

1. Generate $(h_0, h_1) \xleftarrow{\$} \mathcal{R}^2$ both of (odd) weight $|h_0| = |h_1| = w/2$.
2. Compute $h \leftarrow h_1 h_0^{-1}$.
3. Return (h_0, h_1) and h.

Encrypt

- Input: the dense public key h and the message m.
- Output: the ciphertext C.

1. Sample $(e_0, e_1) \in \mathcal{R}^2$ such that $|e_0| + |e_1| = t$.
2. Compute $C = (c_0, c_1) \leftarrow (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1))$.
3. Return C.

Decrypt

- Input: the sparse private key (h_0, h_1) and the ciphertext C.
- Output: the message m or a failure symbol $\perp$.

1. Compute the syndrome $s \leftarrow c_0 h_0$.
2. Compute $\{(e_0', e_1'), \perp\} \leftarrow \texttt{decoder}(s, h_0, h_1)$.
3. If $\perp\ \leftarrow \texttt{decoder}(s, h_0, h_1)$ or $|(e_0', e_1')| \neq t$, output $\perp$ and halt.
4. Else, compute $m \leftarrow c_1 \oplus \mathbf{L}(e_0', e_1')$ and return m.

It is immediate to see that the hybrid PKE we just described is also IND-CPA secure. This follows directly from the IND-CPA security of the KEM used to construct it, which we have proved in Theorem 1.

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[23], 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:

$$Q = \frac{1}{\alpha I^2} \cdot I \cdot C_f = \frac{D_f \cdot C_f}{\alpha \mathsf{MAXDEPTH}}. \tag{3}$$

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

$$Q_{AES-N} = \frac{2^N D_{AES-N} \cdot C_{AES-N}}{\mathsf{MAXDEPTH}}. \tag{4}$$

The proposition follows by comparing (3) with (4). □