INTERNATIONAL JOURNAL OF COMPUTER MATHEMATICS: COMPUTER SYSTEMS THEORY
2021, VOL. 6, NO. 4, 364–374
https://doi.org/10.1080/23799927.2021.1930176

Taylor & Francis
Taylor & Francis Group

On the applicability of the Fujisaki–Okamoto transformation to the BIKE KEM

Nir Drucker[a,b], Shay Gueron[a,b], Dusan Kostic[c] and Edoardo Persichetti [d]

[a]Department of Mathematics, University of Haifa, Haifa, Israel; [b]Amazon Web Services, Seattle, WA, USA; [c]EPFL, Lausanne, Switzerland; [d]Florida Atlantic University, Boca Raton, FL, USA

ABSTRACT

The QC-MDPC code-based KEM BIKE is one of the Round-3 candidates of the NIST PQC standardization project. Its Round-2 specification document described variants claiming to have IND-CCA security. The security proof used the Fujisaki–Okamoto transformation and a decoder targeting a Decoding Failure Rate (DFR) of 2^{-128} (for Level-1 security). However, several aspects needed to be amended in order for the IND-CCA proof to hold. The main issue is that using a decoder with DFR of 2^{-128} does not necessarily imply that the underlying PKE is δ-correct with $\delta = 2^{-128}$, as required. In this paper, we handle the necessary aspects to ensure the security claim is correct. In particular, we close the gap in the proof by defining the notion of *message-agnostic* PKE. We show that the PKEs underlying the BIKE versions are message-agnostic. This implies that BIKE with a decoder that has a sufficiently low DFR is also an IND-CCA KEM.

ARTICLE HISTORY
Received 22 May 2020
Revised 19 February 2021
Accepted 4 May 2021

KEYWORDS
BIKE; post-quantum cryptography; NIST; QC-MDPC codes; Fujisaki–Okamoto

2010 MATHEMATICS SUBJECT CLASSIFICATIONS
94A60; 11T71

1. Introduction

Bit Flipping Key Encapsulation (BIKE) is a code-based Key Encapsulation Mechanism (KEM) built using Quasi-Cyclic Moderate-Density Parity-Check (QC-MDPC) codes, which combined the work of [4,5] into a single submission to the NIST PQC standardization project. The original submission [1] in Round 1 featured three variants named BIKE-1, BIKE-2 and BIKE-3. In the Round 2 specification [2] three KEM additional variants were introduced, named BIKE-1-CCA, BIKE-2-CCA, BIKE-3-CCA, which were claimed to be IND-CCA secure. We have identified several gaps affecting this claim and casting doubts on its validity. The first gap is that the specified decoder was not defined to have a constant number of steps, so it was inherently not a constant-time algorithm. In addition, no constant-time implementation for this decoder was provided. This gap was resolved in [6], together with a study that identified some efficient decoders and their constant-time implementations. The second gap, which was reported in [6], but not yet resolved, was the assumption that δ, as used in the implicit-rejection version of Fujisaki–Okamoto transformation ($FO^{\not\perp}$, as described in [8]) for converting a δ-correct Public Key Encryption (PKE) into an IND-CCA KEM, is the same as the DFR of the decoder specified for BIKE in [2]. Without a proof to this effect, IND-CCA security could not be claimed, even if the decoder presented were to achieve the target DFR (e.g. 2^{-128} for Category-1). Finally, the protocol formulation of BIKE-2-CCA (and BIKE-3-CCA) did not allow a direct application of [8, Theorem 4] before first resolving the δ-correctness gap. We argue that it is easier to slightly modify the flows as described in the text.

CONTACT Edoardo Persichetti epersichetti@fau.edu

The subtle gap between δ and the DFR affects other schemes that have a nonzero decapsulation failure probability, such as LEDAcrypt-PKC [3], based on the McEliece framework, for which, however, a proof is only sketched (see [3, Lemma 1.4.1]). We fully flesh out this idea by defining the notion of *message-agnostic* PKE and applying it to BIKE-1-CCA. LEDAcrypt-KEM [3], instead, solves the gap by adding an Extensible Output Function (XOF). We generalize this notion, showing that all Hybrid Encryption (HE) schemes, such as those underlying BIKE-2-CCA and BIKE-3-CCA, are message-agnostic. We amend the description of the BIKE protocols where necessary. Thus, we are now able to substantiate the claim that the BIKE-$*$-CCA variants actually achieved IND-CCA security, given that there exists a decoder with a constant number of steps and a DFR of the required magnitude.

2. Message-agnostic PKEs

General notation. We denote a protocol failure by $\perp$. Uniform random sampling from a set W is denoted by $w \xleftarrow{\$} W$. For an algorithm A, we denote its output by $out = A()$ if A is deterministic, and by $out \leftarrow A()$ otherwise.

A Public Key Encryption (PKE) scheme is defined over a parameter set, and three spaces: the key space $\mathcal{K}$, the message space $\mathcal{M}$ and the ciphertext space $\mathcal{C}$. It consists of three algorithms:

- a Key Generation algorithm $(sk, pk) \leftarrow Gen()$ that generates a key pair, namely a secret key sk and a public key pk;
- an Encryption algorithm $c \leftarrow Encrypt(pk, m)$ that encrypts a message $m \in \mathcal{M}$ and produces a ciphertext $c \in \mathcal{C}$;
- a Decryption algorithm $m' = Decrypt(sk, c)$ that decrypts $c \in \mathcal{C}$ using sk to either a message $m' \in \mathcal{M}$ upon successful decryption or a failure symbol $\perp$ upon decryption failure.

We define the following probabilities

$$p_1(sk, pk, m) = Pr\left[Decrypt(sk, c) \neq m \mid c \leftarrow Encrypt(pk, m)\right]; \tag{1}$$

$$p_2(sk, pk) = Pr[(sk, pk) \leftarrow Gen()]. \tag{2}$$

For a fixed $(sk, pk) \in \mathcal{K}$, and a fixed $m \in \mathcal{M}$, we say that $p_1(sk, pk, m)$ is the 'per-(key,message) pair decryption failure probability', while $p_2(sk, pk)$ is the probability that $Gen()$ outputs the key pair (sk, pk). We also define the 'overall decryption failure probability' as

$$\begin{aligned} p_3 = Pr[&Decrypt(sk, c) \neq m \mid (sk, pk) \leftarrow Gen(), m \xleftarrow{\$} \mathcal{M}, \\ &c \leftarrow Encrypt(pk, m)], \end{aligned} \tag{3}$$

or in other words:

$$p_3 = \frac{1}{|\mathcal{M}|} \sum_{(sk,pk)\in\mathcal{K}, m\in\mathcal{M}} p_2(sk, pk) \cdot p_1(sk, pk, m) \tag{4}$$

since m is chosen uniformly at random from $\mathcal{M}$.

Definition 2.1 (δ-correct PKE [8]): A given PKE is said to be δ-correct if

$$\mathbb{E}\left[\max_{m\in\mathcal{M}} p_1(sk, pk, m)\right] \leq \delta \tag{5}$$

Remark 2.1: Our discussion revolves around PKEs for which the decryption has a nonzero failure probability (that could depend on the encrypted message and/or on the private key). We note that

PKEs with no failures at all (also known as 'perfectly correct'), can be viewed as a special case with $\delta = 0$.

Definition 2.2 (Message-agnostic PKE): A message-agnostic PKE is a PKE with the following property: the equality

$$p_1(sk, pk, m) = p_1(sk, pk, m') \tag{6}$$

holds for every $(sk, pk) \in \mathcal{K}$ and every $m, m' \in \mathcal{M}$. In other words, for a given key pair, the decryption failure probability is the same for all possible messages.

Proposition 2.3: *Consider a message-agnostic PKE. Then,*

$$p_3 = \mathbb{E}\left[\max_{m \in \mathcal{M}} p_1(sk, pk, m)\right] \tag{7}$$

where the expectation is taken over $(sk, pk) \leftarrow Gen()$.

Proof: We have

$$\begin{aligned}
\mathbb{E}\left[\max_{m \in M} p_1(sk, pk, m)\right] &= \mathbb{E}\left[\frac{1}{|\mathcal{M}|}\sum_{m \in \mathcal{M}} p_1(sk, pk, m)\right] \\
&= \sum_{(sk,pk) \in \mathcal{K}} \frac{1}{|\mathcal{M}|}\sum_{m \in \mathcal{M}} p_1(sk, pk, m) \cdot p_2(sk, pk) \\
&= \frac{1}{|\mathcal{M}|}\sum_{(sk,pk) \in \mathcal{K}, m \in \mathcal{M}} p_1(sk, pk, m) \cdot p_2(sk, pk) \\
&= p_3.
\end{aligned}$$

This concludes the proof. ■

Corollary 2.4: *A message-agnostic PKE is* δ*-correct if* $p_3 \leq \delta$.

3. Hybrid Encryption

The Hybrid Encryption (HE) paradigm [9] (also known as KEM-DEM) constructs PKEs by combining an asymmetric component called Key Encapsulation Mechanism (KEM) with a symmetric component called Data Encapsulation Mechanism (DEM). The latter is usually some sort of authenticated cipher, but for our purposes, we can imagine it consists simply of two deterministic algorithms $Encrypt(pk, m)/Decrypt(sk, c)$. Since the focus of this work is the BIKE KEM, we do not give here more details about DEMs, and move on to define KEMs accurately.

A Key Encapsulation Mechanism (KEM) is defined over a parameter set, and three spaces: the key space $\mathcal{K}$, the ciphertext space $\mathcal{C}$ and the shared string space $\mathcal{S}$. It consists of three algorithms:

- a Key Generation algorithm $(sk, pk) \leftarrow Gen()$ that generates a key pair, namely a secret key sk and a public key pk;
- an Encapsulation algorithm $(c, k) \leftarrow Encaps(pk)$ that generates a bit string[1] $k \in \mathcal{S}$ and encapsulates it in a ciphertext $c \in \mathcal{C}$;
- a Decapsulation $k = Decaps(sk, c)$ that returns $k \in \mathcal{S}$ upon successful decapsulation of $c \in \mathcal{C}$ using sk, or $\perp$ upon failure (if using implicit rejection, decapsulation returns a uniform random value k' instead of $\perp$).

Remark 3.1: Some KEMs use a technique called *implicit rejection*. Here, in case of a failure, decapsulation returns a uniform random value k' instead of $\perp$. The reason is that a KEM is normally used in conjunction with a DEM, which uses the KEM output as symmetric key, and therefore, unless a collision occurs, returning the 'wrong' value k' causes a failure in the DEM decryption process. In this way, an incorrect ciphertext is still rejected in the overall HE scheme, as it would by a traditional KEM, but without revealing information to an adversary.

Definition 3.1 (δ-correct KEM [8]): A KEM is δ-correct if

$$p_4 = Pr\left[Decaps(sk, c) \neq k | (sk, pk) \leftarrow Gen(), (c, k) \leftarrow Encaps(pk)\right] \leq \delta \tag{8}$$

Corollary 3.2: *Consider a KEM built using the $FO^{\not\perp}$ transformation [8] over an underlying message-agnostic PKE. Then $p_4 = p_3$.*

Proof: The $FO^{\not\perp}$ transformation [8] converts a PKE to a KEM by composing the T and the $U^{\not\perp}$ transformations. To be more precise, transformation T converts a δ-correct PKE to a δ_1-correct PKE_1, where δ_1 is a function of the number of random oracle calls, by adding a 'derandomization' step and a 're-encryption' step. In particular, if the number of calls to the random oracle G is q_G, we have $\delta_1(q_G) = q_G \cdot \delta$. Then, transformation $U^{\not\perp}$ converts a δ_1-correct PKE_1 (for $FO^{\not\perp}$, exactly the one resulting from T) to a δ_1-correct KEM by choosing a random plaintext $m \xleftarrow{\$} M$ and deriving the shared secret from it. These two steps do not involve a failure option.

It follows that p_4 is equivalent to

$$\begin{aligned} &Pr[Decrypt_1(sk, c) = \perp \ | (sk, pk) \leftarrow Gen(), \\ &c \leftarrow Encrypt_1(pk, m; G(m)), m \xleftarrow{\$} M]. \end{aligned} \tag{9}$$

In T, $Decrypt_1(sk, c)$ returns $\perp$ when $Decrypt(sk, c)$ returns $\perp$ or when the re-encryption failed, which happens only if $Decrypt(sk, c) \neq m$. Thus, (9) is equivalent to

$$p_3 = Pr[Decrypt(sk, c) \neq m \mid (sk, pk) \leftarrow Gen(), m \xleftarrow{\$} \mathcal{M}, c \leftarrow Encrypt(pk, m)].$$

This concludes the proof. ■

We are now ready to show the main result of this section, that is, that the HE paradigm always yields message-agnostic PKEs. We begin by briefly outlining the framework of an HE scheme.

A Hybrid Encryption (HE) scheme is defined over a parameter set, and three spaces: the key space $\mathcal{K}$, the message space $\mathcal{M}$ and the ciphertext space $\mathcal{C}$. It consists of three algorithms:

- a Key Generation algorithm $(sk, pk) \leftarrow Gen()$, same as that of the KEM;
- an Encryption algorithm $c = (c_0, c_1) \leftarrow Encrypt(pk, m)$, that encrypts a message $m \in \mathcal{M}$ and produces a ciphertext $c \in \mathcal{C}$. The ciphertext is obtained by first computing $(c_0, k) = KEM.Encaps(pk)$ and then $c_1 = DEM.Encrypt(k, m)$.
- a Decryption algorithm $m' = Decrypt(sk, c)$ that decrypts $c \in \mathcal{C}$ using sk to either a message $m' \in \mathcal{M}$ upon successful decryption, or a failure symbol $\perp$ upon decryption failure. Here m' is obtained by first computing $k' = KEM.Decaps(sk, c_0)$ and then $DEM.Decrypt(k', c_1)$.

Proposition 3.3: *The HE scheme described above is message-agnostic.*

Proof: Note that *Decrypt* can fail only in two cases: either $KEM.Decaps(sk, c_0)$ returns the failure symbol $\perp$, or it returns an output key $k' \neq k$. In both cases, the decapsulation algorithm depends only on sk and c_0, and therefore a decryption failure is independent of the message m. ■

Table 1. Hash function (random oracle) domains for the BIKE variants.

BIKE-1-CCA	BIKE-2-CCA	BIKE-3-CCA
$H : \{0,1\}^{2r} \to \{0,1\}^{2r}_{[t]}$	$H : \{0,1\}^{\ell} \to \{0,1\}^{2r}_{[t]}$	$H : \{0,1\}^{\ell} \to \{0,1\}^{3r}_{[t]}$
$K : \{0,1\}^{4r} \to \{0,1\}^{\ell}$	$K : \{0,1\}^{r+2\ell} \to \{0,1\}^{\ell}$	$K : \{0,1\}^{2r+2\ell} \to \{0,1\}^{\ell}$
	$L : \{0,1\}^{2r}_{[t]} \to \{0,1\}^{\ell}$	$L : \{0,1\}^{3r}_{[t]} \to \{0,1\}^{\ell}$

Remark 3.2: Hybrid encryption was, historically, introduced as a tool to design IND-CCA secure PKEs. In fact, the advantage of such a construction is that IND-CCA security for the PKE provably reduces to the IND-CCA security of the KEM and DEM components. In our case, however, we are not interested in this feature, but only in the message-agnostic property, and it is enough for the HE scheme to achieve a weaker security notion (e.g. IND-CPA, which is the case for BIKE). Fortunately, it is trivial to show that, for a KEM-DEM scheme whose KEM component is not IND-CCA secure, the resulting hybrid PKE, while not necessarily IND-CCA secure, does preserve the original notion (e.g. IND-CPA).

4. Applications to BIKE

Let $\mathbb{F}_2$ be the binary finite field and $\mathcal{R}$ be the polynomial ring $\mathbb{F}_2[X]/\langle X^r - 1\rangle$. For every $v \in \mathcal{R}$ denote its Hamming weight by $wt(v)$. Note that every $v \in \mathcal{R}$ with odd weight is also invertible [2]. We write $\{0,1\}^l_{[t]}$ to denote the set of all l-bit strings with Hamming weight t. For BIKE, we set $\mathcal{M} = \mathcal{R}, \mathcal{C} = \mathcal{R}^2$ and $\mathcal{S} = \{0,1\}^{\ell_1}$ (hereafter, $\ell_1 = 256$). The parameters common to all schemes are the integers r, w, t, plus an additional parameter ℓ corresponding to the desired length of the shared key. For example, [2] specifies $r = 11779$, $w = 142$, $t = 134$ for BIKE-1-CCA Level-1. The parameter ℓ is set to $\ell = 256$.

The scheme uses a QC-MDPC decoder. In the case of BIKE-1-CCA and BIKE-2-CCA, a decoder is a procedure `decode`: $\mathcal{R}^3 \longrightarrow \{\mathcal{R}^2, \perp\}$. Both variants can (and do) use the same decoder. For BIKE-3-CCA, the procedure `decode` is slightly tweaked, in order to obtain a so-called 'noisy' decoder which returns an output in $\{\mathcal{R}^3, \perp\}$ instead.

The model for the BIKE schemes uses two functions H and K that are called, loosely, 'hash functions' in [2]. For convenience, we keep this loose notion, but point out that, technically, H and K should be viewed as random oracles over the appropriate domains. Our proofs for BIKE-2-CCA and BIKE-3-CCA modify the original flows by adding another hash function, denoted by L. The domains for these hash functions are summarized in Table 1.

Details and explanations about the subtle differences between the hash function, the random oracles, and the relation to a concrete implementation are provided in Section 5.

4.1. δ-correctness of the BIKE-CCA variants

The BIKE specification document in [2, Section 2.4] defines the DFR of a specific decoder `decode` as 'the probability for the decoder to fail when the input (h_0, h_1, e_0, e_1) is distributed uniformly'. Subsequently, the IND-CCA proofs of the BIKE-CCA variants replace the δ-correctness definition of [8] with the DFR in [2, Section 6.2] as follows:

> the resulting KEM will have the exact same DFR as the underlying cryptosystem.

The authors of [6] have already indicated (see Section 7 in [6]) that there is a subtle difference between the two definitions. Note also that Theorem 4 of [8] states that 'If PKE_1 is δ_1-correct, so is $\text{KEM}^{\not\perp}$ [..]'. However, this does not imply that if a KEM is δ_1-correct then PKE_1 is necessarily δ_1-correct. Corollary 2.3 shows that it is true when the underlying PKE is message-agnostic.

4.2. The underlying PKEs

The BIKE [2] submission states that its IND-CCA proofs rely on applying the $FO^{\not\perp}$ transformation of [8] to the underlying cryptosystems. Note that [8] remarks that 'all our transformations require a PKE scheme (and not a KEM). We view it as an interesting open problem to construct similar transformations that only assume (and yield) KEMs'. In fact this paper offers such a construction and applies it to BIKE-2-CCA and BIKE-3-CCA. As a consequence, to properly analyse the proofs, one first needs to extract the underlying PKEs. Since those are not explicitly described in [2], we do this in this paper, by reversing the $U^{\not\perp}$ and T transformations.

In the next section, we discuss the three BIKE CCA variants, which are presented with all the modifications necessary to fill the gaps. The modifications will then be explained and justified in Section 5. We refer the reader to sections 2.2.1 to 2.2.3 of [2] for the original description of the three schemes.

4.3. BIKE-1-CCA

BIKE-1-CCA consists of the following algorithms.

Key generation. Choose $(h_0, h_1, \sigma_0, \sigma_1) \xleftarrow{\$} \mathcal{R}^4$ with $wt(h_0) = wt(h_1) = w/2$ of odd weight, and choose another polynomial $g \xleftarrow{\$} \mathcal{R}$ of odd weight $wt(g) \approx r/2$. Then, set $pk = (f_0, f_1) = (gh_1, gh_0)$ and $sk = (h_0, h_1, \sigma_0, \sigma_1)$.

Encapsulation. Generate $m \xleftarrow{\$} \mathcal{R}$. Compute $(e_0, e_1) = H(mf_0, mf_1)$, where $wt(e_0) + wt(e_1) = t$. Calculate the ciphertext $c = (c_0, c_1) = (mf_0 + e_0, mf_1 + e_1)$ and the shared secret $k = K(mf_0, mf_1, c)$.

Decapsulation. Compute the syndrome $s = c_0h_0 + c_1h_1$. Calculate $(e'_0, e'_1) \leftarrow \texttt{decode}(s, h_0, h_1)$ and $(e''_0, e''_1) = H(c_0 + e'_0, c_1 + e'_1)$. If the decoder returns $\perp$, or $wt(e'_0) + wt(e'_1) \neq t$, or $(e'_0, e'_1) \neq (e''_0, e''_1)$, then return $k = K(\sigma_0, \sigma_1, c)$. Otherwise, return $k = K(c_0 + e'_0, c_1 + e'_1, c)$.

We now proceed to extract the PKE underlying BIKE-1-CCA, which we denote by $\mathcal{E}$-1. This is essentially a version of the McEliece cryptosystem, instantiated with QC-MDPC codes. Thus, the key generation algorithm of $\mathcal{E}$-1 is the same as in BIKE-1-CCA, with the exception of the elements σ_0 and σ_1 which are added as part of the KEM conversion. The encryption/decryption algorithms of $\mathcal{E}$-1 are:

Encryption. Generate $(e_0, e_1) \xleftarrow{\$} \mathcal{R}^2$, where $wt(e_0) + wt(e_1) = t$. Calculate the ciphertext $c = (c_0, c_1) = (mf_0 + e_0, mf_1 + e_1)$.

Decryption. Compute the syndrome $s = c_0h_0 + c_1h_1$. Calculate $(e'_0, e'_1) \leftarrow \texttt{decode}(s, h_0, h_1)$. If the decoding succeeds and $wt(e'_0) + wt(e'_1) = t$, then return $m = f_0^{-1}(c_0 + e'_0)$; else return $\perp$.

Finally, we prove the following result about $\mathcal{E}$-1.

Proposition 4.1: *$\mathcal{E}$-1 is message-agnostic.*

Proof: The decryption of $\mathcal{E}$-1 can fail to output m in two cases: a) $(e'_0, e'_1) \neq (e_0, e_1)$; b) $\texttt{decode}(s, h_0, h_1)$ returns $\perp$ (i. e., decoding failure occurred). According to the analysis in [2], the probability that Case a) occurs is negligible and we therefore ignore it here. Note that $\texttt{decode}$ depends only on h_0, h_1 and the syndrome, $s = h_0e_0 + h_1e_1$, is independent of m. Therefore, the probability that Case b) occurs is also independent of m. ■

Remark 4.1: The case where $c = (c_0, 0) = (mf_0 + e_0, 0)$ is interesting and deserves to be treated separately. In this case, $s = (mf_0 + e_0)h_0$, and it may seem that the syndrome depends on m. However,

this is not true. In fact, from $c_1 = 0 = mf_1 + e_1$ it follows that $m = f_1^{-1}e_1$. Substituting this equality in s yields

$$s = (f_1^{-1}e_1 f_0 + e_0)$$
$$(g^{-1}gh_0^{-1}h_0h_1e_1 + h_0e_0) = h_1e_1 + h_0e_0$$

The latter expression is independent of m, as expected.

4.4. BIKE-2-CCA

BIKE-2-CCA consists of the following algorithms.

Key generation. Choose $\sigma \xleftarrow{\$} \{0,1\}^\ell$ and $(h_0, h_1) \xleftarrow{\$} \mathcal{R}^2$ with $wt(h_0) = wt(h_1) = w/2$ of odd weight. Then, set $pk = h = h_1h_0^{-1}$ and $sk = (h_0, h_1, \sigma)$.

Encapsulation. Generate $m \xleftarrow{\$} \{0,1\}^\ell$. Compute $(e_0, e_1) = H(m)$, where $wt(e_0) + wt(e_1) = t$. Calculate the ciphertext $c = (c_0, c_1) = (e_0 + e_1h, L(e_0, e_1) \oplus m)$ and the shared secret $k = K(m, c)$.

Decapsulation. Compute the syndrome $s = c_0h_0$. Set $(e'_0, e'_1) \leftarrow \texttt{decode}(s, h_0, h_1)$ and $m' = c_1 \oplus L(e'_0, e'_1)$. If the decoder returns $\perp$, or $wt(e'_0) + wt(e'_1) \neq t$, or $(e'_0, e'_1) \neq H(m')$, then return $k = K(\sigma, c)$. Otherwise, return $k = K(m', c)$.

We now proceed to extract the PKE underlying BIKE-2-CCA, which we denote by $\mathcal{E}$-2. There is a substantial difference here, compared to the previous case of BIKE-1-CCA. In fact, the BIKE-2 KEM was originally designed to follow the Niederreiter framework. However, this approach utilizes the fixed-weight error vector (e_0, e_1) as input for the key derivation, and this could create some security issues: an adversary performing a reaction attack would be able to choose specific error patterns that are potentially more likely to cause a decoding failure. As a consequence, the PKE underlying BIKE-2-CCA is created as an HE scheme, composing a one-time pad (playing the role of the DEM) with the simple IND-CPA KEM[2] described below.

Key generation. Choose $sk = (h_0, h_1) \xleftarrow{\$} \mathcal{R}^2$ with $wt(h_0) = wt(h_1) = w/2$ of odd weight. Then, set $pk = h = h_1h_0^{-1}$ and $sk = (h_0, h_1)$.

Encapsulation. Generate $(e_0, e_1) \xleftarrow{\$} \mathcal{R}^2$, where $wt(e_0) + wt(e_1) = t$. Calculate the ciphertext $c = e_0 + e_1h$ and the shared secret $k = L(e_0, e_1)$. Here, L is a hash function that generates the appropriate value.

Decapsulation. Compute the syndrome $s = ch_0$. Set $(e'_0, e'_1) \leftarrow \texttt{decode}(s, h_0, h_1)$. If the decoder returns $\perp$, or $wt(e'_0) + wt(e'_1) \neq t$, then return $\perp$. Otherwise, return $k = L(e'_0, e'_1)$.

To avoid confusion, relative to the names of the various functions used in the scheme, we chose to use a different notation for the hash function that is used for deriving the shared key in the two KEMs. To this end, we introduced the hash function L. As mentioned above, the KEM and the DEM are combined to obtain the PKE $\mathcal{E}$-2. As in every HE scheme, the key generation algorithm of $\mathcal{E}$-2 is the same as that of the above IND-CPA KEM, and it is thus omitted here. The encryption/decryption algorithms of $\mathcal{E}$-2 are given as follows.

Encryption. Generate $(e_0, e_1) \xleftarrow{\$} \mathcal{R}^2$, where $wt(e_0) + wt(e_1) = t$. Calculate the ciphertext $c = (c_0, c_1) = (e_0 + e_1h, L(e_0, e_1) \oplus m)$.

Decryption. Compute the syndrome $s = c_0 h_0$. Set $(e'_0, e'_1) \leftarrow \texttt{decode}(s, h_0, h_1)$. If the decoding succeeds and $wt(e'_0) + wt(e'_1) = t$, then return $m = c_1 \oplus L(e'_0, e'_1)$; else return $\perp$.

Finally, we prove the following result about $\mathcal{E}$-2.

Proposition 4.2: *$\mathcal{E}$-2 is message-agnostic.*

Proof: This follows immediately from Proposition 3.3. Note that, as in the case for Proposition 4.1, we can deem negligible the probability that $(e'_0, e'_1) \neq (e_0, e_1)$ (which would imply $L(e'_0, e'_1) \neq L(e_0, e_1)$ and thus cause a decryption failure). ■

4.5. BIKE-3-CCA

BIKE-3-CCA consists of the following algorithms.

Key generation. Choose $\sigma \xleftarrow{\$} \{0,1\}^{\ell}$ and $(h_0, h_1) \xleftarrow{\$} \mathcal{R}^2$ with $wt(h_0) = wt(h_1) = w/2$ of odd weight, and choose another polynomial $g \xleftarrow{\$} \mathcal{R}$ of odd weight $wt(g) \approx r/2$. Then, set $pk = (f_0, f_1) = (h_1 + gh_0, g)$ and $sk = (h_0, h_1, \sigma)$.

Encapsulation. Generate $m \xleftarrow{\$} \{0,1\}^{\ell}$. Compute $(e, e_0, e_1) = H(m)$, where $wt(e_0) + wt(e_1) = t$ and $wt(e) = t/2$. Calculate the ciphertext $c = (c_0, c_1, c_2) = (e + e_1 f_0, e_0 + e_1 f_1, L(e, e_0, e_1) \oplus m)$ and the shared secret $k = K(m, c)$.

Decapsulation. Compute the syndrome $s = c_0 + c_1 h_0$. Calculate $(e', e'_0, e'_1) \leftarrow \texttt{decode}(s, h_0, h_1)$ and $m' = c_2 \oplus L(e', e'_0, e'_1)$. If the decoder returns $\perp$, or $wt(e') + wt(e'_0) + wt(e'_1) \neq 3t/2$, or $(e', e'_0, e'_1) \neq H(m')$, then return $k = K(\sigma, c)$. Otherwise, return $k = K(m', c)$.

We now proceed to extract the PKE underlying BIKE-3-CCA, which we denote by $\mathcal{E}$-3. Similarly to the case of BIKE-2-CCA, and for the same reasons, this is also a hybrid PKE. The IND-CPA KEM[3] is described below.

Key generation. Choose $sk = (h_0, h_1) \xleftarrow{\$} \mathcal{R}^2$ with $wt(h_0) = wt(h_1) = w/2$ of odd weight, and choose another polynomial $g \xleftarrow{\$} \mathcal{R}$ of odd weight $wt(g) \approx r/2$. Then, set $pk = (f_0, f_1) = (h_1 + gh_0, g)$ and $sk = (h_0, h_1)$.

Encapsulation. Generate $(e, e_0, e_1) \xleftarrow{\$} \mathcal{R}^3$, where $wt(e_0) + wt(e_1) = t$ and $wt(e) = t/2$. Calculate the ciphertext $c = (c_0, c_1) = (e + e_1 f_0, e_0 + e_1 f_1$ and the shared secret $k = L(e, e_0, e_1)$. Here, L is a hash function that generates the appropriate value.

Decapsulation. Compute the syndrome $s = ch_0$. Compute $(e', e'_0, e'_1) \leftarrow \texttt{decode}\ (s, h_0, h_1)$. If the decoder returns $\perp$, or $wt(e') + wt(e'_0) + wt(e'_1) \neq 3t/2$, then return $\perp$. Otherwise, return $k = L(e', e'_0, e'_1)$.

As before, to avoid confusion, we use L to denote the hash function that derives the shared key in the above KEM. We now proceed to combine the KEM and the DEM to obtain the PKE $\mathcal{E}$-3. The key generation algorithm of $\mathcal{E}$-3 is the same as that of the above IND-CPA KEM, while the encryption/decryption algorithms of $\mathcal{E}$-3 are given by:

Encryption. Generate $(e, e_0, e_1) \xleftarrow{\$} \mathcal{R}^3$, where $wt(e_0) + wt(e_1) = t$ and $wt(e) = t/2$. Calculate the ciphertext $c = (c_0, c_1, c_2) = (e + e_1 f_0, e_0 + e_1 f_1, L(e, e_0, e_1) \oplus m)$.

Decryption. Compute the syndrome $s = c_0 + c_1 h_0$. Calculate $(e', e'_0, e'_1) \leftarrow \texttt{decode}(s, h_0, h_1)$. If the decoding succeeds, $wt(e'_0) + wt(e'_1) = t$ and $wt(e') = t/2$, then return $m = c_2 \oplus L(e', e'_0, e'_1)$; else return $\perp$.

Finally, we prove the following result about $\mathcal{E}$-3.

Proposition 4.3: *$\mathcal{E}$-3 is message-agnostic.*

Proof: This follows immediately from Proposition 3.3. Note that, similar to the case of Proposition 4.1 and Proposition 4.2, we can deem negligible the probability that $(e', e'_0, e'_1) \neq (e, e_0, e_1)$ (which would imply $L(e', e'_0, e'_1) \neq L(e, e_0, e_1)$ and thus cause a decryption failure). ■

5. Protocol modifications

Our definition of the BIKE-2-CCA and BIKE-3-CCA protocols is slightly different from the description given in the specification document [2]. This section summarizes the modifications we made to the protocols, and explains why they were necessary.

5.1. Key derivation

One of the crucial steps in the $FO^{\not\perp}$ transformation, and a fundamental part of the proof of [8, Theorem 4], is that the shared key generated from the KEM is obtained by applying a random oracle to the pair (message, ciphertext) of the underlying PKE. However, this is not the case for BIKE. In fact, for BIKE-2-CCA, the algorithm specified in [2] derives the shared key as $k = K(e_0, e_1, c)$. Similarly, for BIKE-3-CCA, the algorithm specified in [2] derives the shared key as $k = K(e, e_0, e_1, c)$. In both cases, this corresponds, effectively, to deriving the key from the randomness of the PKE, rather than from the message. Now, by means of transformation T, this randomness is generated from the message using a dedicated random oracle. It follows that a substantial modification in the proof would be necessary for it to hold for BIKE-2-CCA and BIKE-1-CCA. Such a discrepancy is not discussed nor justified in the BIKE specification document. In the end, we chose to deal with this issue by slightly modifying the encapsulation (and decapsulation) flows.

Our description derives the shared key as $k = K(m, c)$ for both BIKE-2-CCA and BIKE-3-CCA. This modification has two advantages: a) It is consistent with the description of [8], which means the security argument can be applied to BIKE directly; b) It is more efficient since the input to the key derivation function is much shorter; in fact, this now consists only of the $\ell = 256$ bits of the string m instead of the $t \log_2 r$ bits of (e_0, e_1) (for BIKE-2-CCA) or the $3t/2 \log_2 r$ bits of (e, e_0, e_1) (for BIKE-3-CCA).

Note that, technically, the description of BIKE-1-CCA is also not exactly following the blueprint given in [8]. In fact, the scheme derives the shared key as $k = K(mf_0, mf_1, c)$, effectively replacing the message m with (mf_0, mf_1). This vector corresponds to the codeword associated to the message m in the quasi-cyclic code defined by the generator matrix (f_0, f_1). The modification, however, is much less significant in this case: there is a one-to-one correspondence between messages and codewords in a linear code (for a fixed choice of generator matrix), and no random oracle is involved in the process. Furthermore, as mentioned in [2], this is a convenient choice. In fact, unlike the case of BIKE-2-CCA and BIKE-3-CCA, where m is obtained immediately by 'undoing' the one-time pad, for BIKE-1-CCA one would need to recover m from (mf_0, mf_1). This requires performing a polynomial inversion, which is one of the most expensive operations, and one that BIKE-1 explicitly aimed to avoid. Therefore, in the end, we chose to leave the description of BIKE-1-CCA as in [2].

5.2. Random oracles

The encapsulation algorithm in [2] uses the same function K for the encapsulation step, $c = (c_0, c_1) = (e_0 + e_1 h, K(e_0, e_1) \oplus m)$, and for the key-derivation step, $k = K(e_0, e_1, c)$. Our definition, instead,

uses an independent function L for $c = (c_0, c_1) = (e_0 + e_1 h, L(e_0, e_1) \oplus m)$ since this is, formally, a different random oracle (it is, in fact, the key-derivation function in the underlying IND-CPA KEM). Similarly, our definition of BIKE-3-CCA is also different from that of [2]. Encapsulation in [2] uses the same function K for the encapsulation step $c = (c_0, c_1, c_2) = (e + e_1 f_0, e_0 + e_1 f_1, K(e, e_0, e_1) \oplus m)$ and for the key-derivation step $k = K(e, e_0, e_1, c)$. Once again, in our definition we use an independent function L for $c = (c_0, c_1, c_2) = (e + e_1 f_0, e_0 + e_1 f_1, L(e, e_0, e_1) \oplus m)$. Obviously, the decapsulation procedure is modified accordingly for both BIKE-2-CCA and BIKE-3-CCA.

For our model, we consider the functions H, K and L as being chosen uniformly at random from the set of all functions with the associated domains and the required constraints on their range, as defined in Table 1. The new model definition does not affect the way that BIKE-2-CCA and BIKE-3-CCA can be (and are) instantiated in practice (see [2] for details). Indeed, K and H are implemented by first 'extracting' the input with a hash function X (e.g. SHA-384) and subsequently 'expanding' the hash digest to an output of the desired length (and constraints), using some pseudorandom generator. This can be done for the new function L as well. Since the inputs to K, H and L have different lengths, this already provides domain separation when applying the extraction step X. For the instantiations, we assume that the SHA-384 digests are indistinguishable from random strings (of the proper lengths and with the required constraints) by an observing adversary with a given number of queries and that the distinguishing advantage from the pseudorandom expansion is also negligible here.

6. Conclusion

In this work, we formalized the notion of message-agnostic PKE, and proved that is always satisfied for PKEs obtained via the KEM-DEM framework. As a consequence, we showed that constructing a decoder with the required DFR is sufficient for supporting the IND-CCA claim of BIKE. Together with [6], this completes the missing parts of the IND-CCA proof given in [2].

Some minor modifications were necessary for our analysis to fit the cases of BIKE-2-CCA and BIKE-3-CCA. The subtlety lies in the difference between the PKEs underlying the three variants. For BIKE-1-CCA, this is essentially McEliece, where the randomness (error vector) is already independent of the message (codeword). On the other hand, BIKE-2-CCA and BIKE-3-CCA are based on the Niederreiter and Ouroboros cryptosystems, respectively. Here, the message was originally conveyed in the error vector. As a consequence, it was necessary to use a different approach, building a PKE by means of the HE paradigm. This is obtained by combining the original IND-CPA KEM (BIKE-2 and BIKE-3, respectively) with a simple DEM (one-time pad), so that the message (a fixed-length bit string) is effectively encrypted using the KEM output (a hash of the error vector) as key. In the end, the small modification detailed in Section 5 makes the underlying PKE message-agnostic, and extends the conclusion to BIKE-2-CCA and BIKE-3-CCA as well.

Of course, to complete the proof, the BIKE specification needs to define at least one decoder (or multiple options for a decoder), `decode`, associated with the scheme and the chosen parameters, that provably provides the appropriate DFR. To summarize, even after closing the δ-correctness gap, the IND-CCA proof still needs to rely on the three assumptions:

(1) `decode` has a DFR of 2^{-128} (for Level-1).
(2) `decode` can be implemented in constant-time.
(3) The probability for a decoding success but a decryption failure is negligible.

Some examples for efficient and constant-time constructions for `decode`, their suggested parameters, and *extrapolated* DFR values, are explored in [6,7]. These extrapolations (perhaps tuned with parameters that provide comfortable margins) may be perceived as enough practical evidence for a sufficiently low DFR.

We remark that our solution is not the only valid one; for example, LEDAcrypt [3] independently and concurrently proposed to use a XOF for the same purpose, following a different construction

(based on transformation $U_m^{\not\perp}$ of [8]). Compared to LEDAcrypt's choice, our formulation is slightly more generic, as it can be instantiated with any hash function (not necessarily having variable output length). Our solution may also present some computational advantages, although that is not the focus of this work, and would deserve further investigation.

Notes

1. Generally, this is used as a shared key for a symmetric encryption scheme.
2. This corresponds to the BIKE-2 variant described in [2].
3. This corresponds to the BIKE-3 variant described in [2].

Acknowledgments

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.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This research was supported by: NSF-BSF (United States-Israel Binational Science Foundation) [grant number 2018640]; NSF Grant CNS (Division of Computer and Network Systems) [grant number 1906360]; The Israel Science Foundation [grant number 3380/19].

ORCID

Edoardo Persichetti http://orcid.org/0000-0002-1895-377X

References

[1] N. Aragon, P.S.L.M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C.A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.P. Tillich, V. Vasseur, and G. Zémor, *BIKE: Bit Flipping key encapsulation* (2017). Available at https://bikesuite.org/files/BIKE.2017.11.30.pdf.

[2] N. Aragon, P.S.L.M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C.A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.P. Tillich, V. Vasseur, and G. Zémor, *BIKE: Bit flipping key encapsulation* (2017). Available at https://bikesuite.org/files/round2/spec/BIKE-Spec-2019.06.30.1.pdf.

[3] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, *LEDAcrypt: Low-dEnsity parity-check coDe-bAsed cryptographic systems* (2020). Available at https://www.ledacrypt.org/.

[4] P.S. Barreto, S. Gueron, T. Gueneysu, R. Misoczki, E. Persichetti, N. Sendrier, and J.P. Tillich, *CAKE: Code-based algorithm for key encapsulation*, IMA International Conference on Cryptography and Coding, Springer, 2017, pp. 207–226.

[5] J.C. Deneuville, P. Gaborit, and G. Zémor, *Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory*, International Workshop on Post-Quantum Cryptography, Springer, 2017, pp. 18–34.

[6] N. Drucker, S. Gueron, and D. Kostic, *On constant-time QC-MDPC decoding with negligible failure rate*, Cryptology ePrint Archive, Report 2019/1289 (2019), Available at https://eprint.iacr.org/2019/1289.

[7] N. Drucker, S. Gueron, and D. Kostic, *QC-MDPC decoders with several shades of gray*, Cryptology ePrint Archive, Report 2019/1423 (2020). Available at https://eprint.iacr.org/2019/1423.

[8] D. Hofheinz, K. Hövelmanns, and E. Kiltz, *A modular analysis of the Fujisaki-Okamoto transformation*, in *Theory of Cryptography*, Y. Kalai and L. Reyzin, eds., Springer International Publishing, Cham, 2017, pp. 341–371. doi:10.1007/978-3-319-70500-2_12.

[9] V. Shoup, *Using hash functions as a hedge against chosen ciphertext attack*, in International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2000, pp. 275–288.