BIKE Crash: Security Analysis of BIKE-1, A Post-Quantum Key Encapsulation Mechanism

Carter Burn, Varad Deshpande

December 10, 2018

Abstract

The Round-1 submissions for NIST's post-quantum cryptography standardization process have been posted. BIKE is one of the proposed candidates which provides a construction for a key encapsulation mechanism (KEM). BIKE makes use of Quasi Cyclic Moderate Density Parity Check Codes in its construction. The authors of BIKE provide three variants for KEM: BIKE-1, BIKE-2 and BIKE-3. In this paper, we examine BIKE-1 which is based on the McEliece cryptosystem. Particularly, we perform a security analysis for BIKE-1 against four types of attacks: Chosen Plaintext Attacks, IND-Chosen Ciphertext Attacks, Side channel/Reaction Attacks and Fault Attacks. Through our analysis, we show that BIKE-1 is does not provide IND-CCA security and is also vulnerable to fault attacks. Finally, we discuss a possible modification for the scheme that can make the scheme IND-CCA secure against our suggested chosen ciphertext attack and why this modification does not apply to the general case at this moment.

1 Introduction

One of the unanswered questions in cryptography is: "what is the plan in the post-quantum world?" The National Institute of Standards and Technology seeks to answer this question using a rigorous standardization process, similar to how AES was standardized in the early 2000s. The NIST post-quantum cryptography standardization process began with a submissions deadline on November 30, 2017. Following the submissions, NIST enters into nearly two years of analysis to choose the standard for post-quantum cryptography. Currently, the process is in Round 1, where 69 of the original submissions are being analyzed to move on to Round 2, expecting to begin in 2019. One of those submissions is the Bit Flipping Key Encapsulation (BIKE) scheme. BIKE uses linear codes, specifically Quasi Cyclic Moderate Density Parity Check (QC-MDPC) codes, to encapsulate a symmetric key between two parties to further communicate using symmetric key cryptography schemes. In this work, we perform a security analysis of BIKE-1 (a BIKE variant based on the McEliece cryptosystem) that considers security against chosen plaintext attacks, chosen ciphertext attacks, reaction or side-channel attacks, and fault attacks. Specifically, we demonstrate a chosen ciphertext attack and a fault attack on BIKE-1. These findings suggest that BIKE-1 may need some further modifications in order to be considered a

strong candidate for a post-quantum standard. In this paper, we begin by presenting necessary background with linear codes and QC-MDPC codes as well as cryptosystems that BIKE uses as its foundation. Then, we present the three key encapsulation algorithms used by BIKE and discuss the security parameters used for BIKE. Next, we discuss the security analysis of BIKE and show BIKE's security against chosen plaintext attacks and GJS reaction attacks, while exhibiting a chosen ciphertext attack and fault attack against BIKE. We also discuss a possible modification to BIKE in order to protect it against chosen ciphertext attacks and why this modification might not possible at this current stage. Finally, we present further extensions of this work to be completed in the future.

2 Background And Related Work

First, we present the necessary background needed to understand BIKE [1]. BIKE, like many post-quantum cryptography schemes, uses linear codes as its basis. 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 $\mathbb{F}_2^n$. A matrix $G \in \mathbb{F}_2^{k \times n}$ is known as the *generator matrix* of the code $\mathcal{C}$ if and only 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 the code $\mathcal{C}$ if and only if $\mathcal{C} = \{c \in \mathbb{F}_2^n \mid Hc^T = 0\}$. With these definitions, we can fully define how linear codes are constructed. A *codeword* $c \in \mathcal{C}$ of a vector $m \in \mathbb{F}_2^{(n-r)}$ is computed as $c = mG$. A vector $s \in \mathbb{F}_2^n$ computed as $s^T = He^T$ is know as the syndrome of e. The original intention of linear codes was to correct errors when transmitting a message across a noisy channel. When one party wants to send a message over a noisy channel, they compute the codeword of their message and send the codeword. The receiving party will then compute the syndrome of the received codeword. The syndrome is equal to the 0 vector if there were no errors in the received message. Otherwise, the receiving party will have to use a decoding algorithm to extract the errors from the received codeword. We will discuss a relevant decoding algorithm later in this work. One additional definition that is useful is the Hamming weight of a vector v denoted as $|v|$. The Hamming weight is simply the number of elements in the vector that are not equal to 0.

Building off of the definition of binary codes, we present quasi-cyclic codes (QC codes). A (n, k)-linear code is QC if there is some integer n_0 such that every cyclic shift of a codeword by n_0 positions is also a codeword. In the case that $n = n_0 r$ for some integer r, we can have both the generator and the parity check matrices composed of $r \times r$ circulant blocks concatenated together. One convenient property of the $r \times r$ circulant matrices is that there exists a natural ring isomorphism between the binary $r \times r$ circulant matrices and the quotient polynomial ring $\mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$. In particular the circulant matrix A whose first row is the vector $(a_0, \ldots, a_{r-1})$ is mapped to the polynomial in $\mathcal{R}$ as: $a_0 + a_1 X + \cdots + a_{r-1} X^{r-1}$ (the first row is the coefficient vector for the polynomial). We can expand QC codes to QC Moderate Density Parity Check Codes (QC-MDPC). The main extension from normal QC codes is that another parameter w defines the constant row weight of the parity check matrix. In order to generate a (n, r, w)-QC-MDPC code with $n = n_0 r$, we select the first

rows $h_0, \ldots, h_{n_0-1}$ of the parity-check matrix blocks $H_0, \ldots, H_{n_0-1}$ with weight $\sum_{i=0}^{n_0-1} |h_i| \leq w$ uniformly at random. The remaining rows are then obtained with $r-1$ quasi-cyclic shifts of each of the rows $h_0, \ldots, h_{n_0-1}$. This construction will allow for the easy generation of secure linear coding cryptographic schemes with practical key sizes.

With these mathematical foundations, we can now explain construction code based cryptographic schemes. The McEliece Cryptosystem [2] is one such cryptographic scheme. The original system used binary Goppa codes, which resulted in impractically long keys. The McEliece system based its security on "Syndrome Decoding", which happens to be a NP-hard problem. The Syndrome Decoding problem is as follows: Given s, H, t, find e with Hamming weight $|e| \leq t$ such that $s^T = eH^T$. In order to overcome the problem of long keys, Misoczki et al.[3] suggested constructing the McEliece cryptosystem using QC-MDPC codes. The QC-MDPC varian of the McEliece scheme is able to provide the same level of security with significantly shorter key sizes. The three phases of the McEliece scheme based on QC-MDPC codes are: Key Generation, Encryption, and Decryption.

- **Key Generation** $Gen(\cdot)$: Takes as input n (length of the code $\mathcal{C}$), r (co-dimension of $\mathcal{C}$), t (number of errors that $\mathcal{C}$ can correct), and w (constant row weight in the parity-check matrix). The function outputs the generator matrix of $\mathcal{C}$, $G \in \mathbb{F}_2^{(n-r)\times n}$ as the public key and the parity-check matrix of $\mathcal{C}$, $H \in \mathbb{F}_2^{r\times n}$ as the private key. Generally, G is written in a standardized form given by $G = [I|Q]$, where I is a $(n-r) \times (n-r)$ identity matrix and Q is obtained from the blocks of H. The parity check matrix is chosen as described above.

- **Encryption** $Enc(\cdot)$: Takes as input the public key G and a message $m \in \mathbb{F}_2^{(n-r)}$. The function will generate at random an error vector $e \in \mathbb{F}_2^n$ of weight t and compute: $c = mG + e$ and return c.

- **Decryption** $Dec(\cdot)$: Takes as input the private key H and the encrypted message $c \in \mathbb{F}_2^n$. Then, calls a decoding algorithm (in this case, the Bit Flipping Algorithm) to decode the syndrome to give $c' = mG$. Since G is in the standard form the original message can be extracted from the first $n-r$ bits of c'

The decoding algorithm used be BIKE and QC-MDPC McEliece is the Bit Flipping Algorithm. The intuition behind this decoding algorithm is that we can flip the bits of the ciphertext in order to make the syndrome zero (no errors). Once the syndrome is the zero vector, we have found the ciphertext that makes the syndrome zero, which is equivalent to the error vector that drives the syndrome to the zero vector. The algorithm takes advantage of the

following reduction of the syndrome:

$$
\begin{aligned}
s &= c \cdot H^T && (1)\\
&= (mG + e) \cdot H^T && (2)\\
&= mGH^T + e \cdot H^T && (3)\\
&= e \cdot H^T && (4)\\
& && (5)
\end{aligned}
$$

This is because $GH^T = 0$. Thus, computing the syndrome of the ciphertext is equivalent to computing the syndrome of the error vector. This means flipping the bits of the ciphertext is equivalent to flipping the corresponding bits in the error vector. This essentially means that if one flips the bits of the ciphertext (and changes the ciphertext) causing the syndrome to become a zero vector, then we have corrected all the errors present in the ciphertext. The psuedocode for the Bit Flipping Algorithm is:

Input: $c = mG + e$, limit
Output: Message : mG or Decoding Failure
Compute syndrome $s = cH^T$;
for $i = 0 \rightarrow limit$ **do**
  for *every codeword bit* j **do**
    Count unsatisfied parity-check equations $\#_{upc}$;
  end
  Set $b = Max_{upc}$;
  for *every codeword bit* j **do**
    if $\#_{upc} \geq b$ **then**
      Flip codeword bit c_j ;
      Update syndrome $s = s \oplus h_j$;
    end
  end
  if $s = 0$ **then**
    return c
  end
end
return *Decoding Failure;*

Algorithm 1: Bit Flipping Algorithm

The final piece of background necessary prior to the introduction of BIKE is the definition of a Key Encapsulation Method (KEM). A KEM consists of three algorithms: Gen, Encaps, and Decaps. The purpose of a KEM is to provide two parties the ability to exchange symmetric keys using public key cryptography. Note that the purpose of a KEM is to not have the ability to encrypt messages, but rather to exchange a symmetric key to be used in a chosen symmetric cryptography scheme for encrypting further messages. [1]

3 BIKE-1: Key Encapsulation Mechanism

3.1 Security Definitions

BIKE is first initialized with a chosen security parameter, λ. λ is chosen based on its relation to the hardness of two problems: $(2,1)$-QC Syndrome Decoding and $(2,1)$-QC Codeword Finding. The $(2,1)$-QC Syndrome Decoding Problem (QCSD) is: Given s, H, t for a $(2,1)$-QC code $(n_0 = 2, k_0 = 1)$), find e such that $|e| \leq t$ and $s^T = eH^T$. The $(2,1)$-QC Codeword Finding (QCCF) Problem is: Given H, t, find a codeword c such that $|c| = t$ and $cH^T = 0$. Both of these problems are NP-hard and the correct selection of the block size and weight t will provide a certain level of hardness against an adversary attempting to find e and c for each problem. For the QCSD Problem, we set the block size to r and weight to t. For the QCCF Problem, we set the block size also to r and the weight to w. When r, t, w are selected to make these problems hard, we define $\lambda \approx t - \frac{1}{2}\log(r) \approx w - \log(r)$. Thus, λ is chosen based on hardness assumptions for the QCSD and QCCF problems. There are certain constraints placed on the choice of r. Specifically, r must be large enough to decode t errors with negligible probability and 2 must be primitive modulo r, which forces r to be a prime number and implies that $(X^r - 1)$ has two irreducible factors (one of which is $(X^r - 1)$). With these parameters, the computational cost of decoding t errors with a uniformly chosen binary linear code of length $n = 2r$ and dimension r is $\Omega(\exp(\lambda))$. Finally, BIKE assumes the use of a cryptographically secure hash function $\mathbf{K} : \{0,1\}^n \to \{0,1\}^{l_K}$ where l_K is the desired symmetric key length for the encapsulated key. [1]

3.2 BIKE-1 KEM

In this section, we describe BIKE's KEM implementation using the security definitions. To review, a KEM has three algorithms: Gen (which outputs a public and private key pair), Encaps (which outputs some ciphertext and the encapsulated key), Decaps (which outputs the encapsulated key).

Gen

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

0. Set parameters r, w based on λ (such that $\lambda \approx w - \log(r)$)
1. Generate $h_0, h_1 \in_R \mathcal{R}$ of odd weight such that $|h_0| = |h_1| = \frac{w}{2}$
2. Generate $g \in_R \mathcal{R}$ of odd weight such that $|g| \approx \frac{r}{2}$
3. Compute $(f_0, f_1) \leftarrow (gh_1, gh_0)$

Recall that $\mathcal{R}$ represents the quotient polynomial ring $\mathbb{F}_2[X]/(X^r - 1)$ that is an isomorphism from $r \times r$ circulant block matrices.

Encaps

– Input: (f_0, f_1) (public key)

– Output: Encapsulated key K and ciphertext c

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

DECAPS

– Input: (h_0, h_1) (private key) and ciphertext c

– Output: Decapsulated key K or failure symbol $\perp$

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

4 Formal Security Analysis For BIKE-1

4.1 Indistinguishibility Against Chosen Plaintext Attacks

In the paper, the authors show that BIKE-1 is indistinguishable against chosen plaintext attacks. In order to demonstrate this, they give the following definition of what it means for a scheme to be IND-CPA secure:

IND-CPA Security: A key encapsulation mechanism is IND-CPA (passively) secure if the outputs of the following two games are computationally indistinguishable.

G_{real}	G_{fake}
$(sk, pk) \leftarrow Gen(\lambda)$	$(sk, pk) \leftarrow Gen(\lambda)$
$(c, K) \leftarrow$ ENCAPS(pk)	$(c, K) \leftarrow$ ENCAPS(pk)
	$K^* \xleftarrow{\$} \mathcal{K}$
Output:(pp,pk,c,K)	Output:(pp,pk,c,K*)

The authors state and prove that all three variants of BIKE are IND-CPA secure in the Random Oracle Model under (2,1)-QCCF, (2,1)-QCSD and (3,1)-QCSD assumptions. For the complete proof, please refer to [1]. Here we only restate the proof for BIKE-1.

Proof: Here, the hash function $\mathbf{K}$ is modeled as a random oracle. We need to show that if an adversary is able to distinguish one game from another, then this fact can be exploited to break one or more of the problems above

in polynomial time. Let us denote the probabilistic polynomial time adversary against the IND-CPA game for BIKE-1 by $\mathcal{A}$. In the paper, the authors define the following games where $\mathcal{A}$ receives the encapsulation at the end of each game.

Game G_1: This corresponds to an honest run of the protocol, and is the same as Game G_{real}. In particular, the simulator has access to all keys and randomness.

Game G_2: In this game, the simulator picks uniformly at random the public key, specifically (f_0, f_1). The rest of the game then proceeds honestly.

An adversary that is able to distinguish between these two games is therefore able to distinguish between a well-formed public key and a randomly-generated one. Here, the public key in G_1 corresponds to a valid (2,1)-QCCF instance for BIKE-1, while it is random in G_2. Thus, we can define the advantage for the adversary $\mathcal{A}$ as follows:

$$\mathrm{Adv}^{G_1-G_2}(\mathcal{A}) \leq \mathrm{Adv}^{(2,1)-QCCF}(\mathcal{A}')$$

where $\mathcal{A}'$ is a polynomial time adversary for the (2,1)-QCCF problem.

Game G_3: Now, the simulator also picks uniformly at random the ciphertext (c_0, c_1). The encapsulated key K is still generated honestly.

If an adversary is able to distinguish the game G_2 from the game G_3, then it can solve one of the QCSD problems. Therefore, we get the advantage of the adversary as,

$$\mathrm{Adv}^{G_2-G_3}(\mathcal{A}) \leq \mathrm{Adv}^{(2,1)-QCSD}(\mathcal{A}")$$

Game G_4: In this game, the true value of K is replaced with a uniformly random value K^*. Since $\mathbf{K}$ is modeled as a random oracle, its output is pseudo-random, and an adversary only has negligible advantage ϵ. Thus, the advantage of the adversary is,

$$\mathrm{Adv}^{G_3-G_4}(\mathcal{A}) \leq \epsilon$$

Therefore, for BIKE-1 we have,

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

This shows that given the security assumptions, BIKE-1 is IND-CPA secure.

4.2 Indistinguishibility Against Chosen Ciphertext Attacks

The authors of BIKE noted that it is difficult to achieve Chosen Ciphertext Security for a scheme using the Bit Flipping decoding technique. Here, we present a IND-CCA attack on BIKE-1 in order to prove that BIKE-1 does not achieve IND-CCA security.

The game followed for the CCA attack is similar to the game as defined for IND-CPA security. We first define two simulations of the protocol: G_{real} and G_{fake}:

G_{real} is an honest run of the protocol where G_{fake} is an honest run, except K is replaced by uniformly sampling a value from the output space of the hash algorithm ($\{0,1\}^{l_K}$). For the CCA game, we define an adversary $\mathcal{A}$, who has

G_{real}	G_{fake}
$(sk, pk) \leftarrow \text{GEN}(\lambda)$	$(sk, pk) \leftarrow \text{GEN}(\lambda)$
$(c, K) \leftarrow \text{ENCAPS}(pk)$	$(c, K) \leftarrow \text{ENCAPS}(pk)$
	$K^* \xleftarrow{\$} \mathcal{K}$
Output: (pk, c, K)	Output: (pk, c, K^*)

access to ENCAPS and DECAPS as oracles. Thus, $\mathcal{A}$ can request key encapsulations given a public key (f_0, f_1) and $\mathcal{A}$ can request key decapsulations for any ciphertext c that $\mathcal{A}$ wants decapsulated as long as the requested decapsulation is not the same ciphertext as provided in the game. Let G_0 be a run of G_{real} and G_1 be a run of G_{fake}. Formally, the experiment is defined such that a verifier V selects at random $b \in \{0, 1\}$ and produces output of G_b and sends the tuple $T = (pk, c, K')$ to $\mathcal{A}$. $\mathcal{A}$ must output $b' \in \{0, 1\}$, selecting that T is output from $G_{b'}$. $\mathcal{A}$ succeeds if $b' = b$. For IND-CCA security, the probability that $\mathcal{A}$ succeeds must be: $\Pr(\mathcal{A} \text{ succeeds}) \leq \frac{1}{2} + \epsilon(\lambda)$. Our attack will show that $\Pr(\mathcal{A} \text{ succeeds}) \geq \frac{1}{2} + \epsilon(\lambda)$.
Given the tuple $T = (pk, c, K')$, we have the following information:

$$\begin{aligned} c &= (c_0, c_1) \\ c_0 &= mf_0 + e_0 \\ c_1 &= mf_1 + e_1 \\ pk &= (f_0, f_1) \end{aligned}$$

$\mathcal{A}$ constructs a new ciphertext c' as follows:

$$\begin{aligned} c' &= (c'_0, c'_1) \\ c'_0 &= mf_0 + e_0 + f_0 = (m+1)f_0 + e_0 \\ c'_1 &- mf_1 + e_1 + f_1 = (m+1)f_1 + e_1 \end{aligned}$$

$\mathcal{A}$ simply added the corresponding public key (f_0, f_1) to the ciphertext. Since this is a different ciphertext than c, it is a valid ciphertext that can be used in a query to DECAPS. $\mathcal{A}$ then queries $\text{DECAPS}(c')$. We claim the syndrome of c is the same as the syndrome of the newly computed ciphertext c':

$$\begin{aligned} s &= c_0 h_0 + c_1 h_1 \\ &= (mf_0 + e_0)h_0 + (mf_1 + e_1)h_1 \\ &= (mf_0 h_0 + mf_1 h_1) + e_0 h_0 + e_1 h_1 \\ &= (mgh_1 h_0 + mgh_0 h_1) + e_0 h_0 + e_1 h_1 \\ &= e_0 h_0 + e_1 h_1 \end{aligned}$$

$$
\begin{aligned}
s' &= c'_0 h_0 + c'_1 h_1 \\
&= ((m+1)f_0 + e_0)h_0 + ((m+1)f_1 + e_1)h_1 \\
&= ((m+1)f_0 h_0 + (m+1)f_1 h_1) + e_0 h_0 + e_1 h_1 \\
&= ((m+1)gh_1 h_0 + (m+1)gh_0 h_1) + e_0 h_0 + e_1 h_1 \\
&= e_0 h_0 + e_1 h_1
\end{aligned}
$$

Therefore, c and c' will both exhibit the same decoding behavior as the syndromes of c and c' are identical. Therefore, $\mathcal{A}$ can compare the value outputted by $\textsc{Decaps}(c')$ with K' from T to find the correct value of b. If $\textsc{Decaps}(c') = K'$, $\mathcal{A}$ sets $b' = 0$ (T was an instance of G_0, or a real run of the protocol). Otherwise $\mathcal{A}$ outputs $b' = 1$. One important note is the case that $\textsc{Decaps}$ outputs $\perp$. Given that the parameters for BIKE are chosen correctly (which is the case when $\textsc{Gen}(\lambda)$ is ran), the probability of a decoding error is very low. In this event, $\mathcal{A}$ does not succeed, however since this happens with low probability, the probability that $\mathcal{A}$ succeeds is high. Thus, $\Pr(\mathcal{A} \text{ succeeds}) > \frac{1}{2} + \epsilon(\lambda)$, which proves BIKE-1 is not CCA secure.

4.3 GJS Reaction Attack

The GJS reaction attack aims to recover the private key h by exploiting the information obtained from decoding failures. It does so by creating specific error patterns and querying the decoder. By repeating the process for multiple such patterns and observing the output of the decoder in each case, the adversary can obtain a distance spectrum which allows him to reconstruct the private key h. For a detailed description of the attack, please refer to [6].

All three variants of BIKE make use of a ephemeral KEM key pair for each transaction, i.e, a KEM key generation is performed for each key exchange. As a result, an adversary has only a single opportunity to observe the decryption for a given key pair. Hence, the GJS attack is defeated as the adversary now is unable to create any statistical information for different error patterns which is necessary in order to construct the distance spectrum.

The authors note that in order to make the scheme more efficient, the initiator may want to precompute the KEM key pairs before engaging in key exchange sessions. The two parties must also take care to follow policies that allow for secure storage of the pre-generated KEM key pairs or a reaction attack could be introduced into the system.

4.4 Fault Attacks

Fault attacks are a class of attacks that allow the adversary to tamper with the hardware circuit which is used to implement the cryptographic protocol. There are many novel ways in which fault attacks can be constructed in order to extract the secret information. In this section, we propose two types of fault attacks. The first attack we suggest allows us to recover the symmetric key. The second attack allows us to cause obstruction in the operation of the scheme

without letting adversary become aware of the attackers presence. The attacks that we propose make use of single event upsets (SEU) and voltage spikes and glitches. [5] defines SEUs and voltage glitches as follows:

Single Event Upset: SEUs are flips in a cell's logical state to a complementary state. The transition can be temporary, if the fault is produced in a dynamic system, or permanent if it appears in a static system.

Voltage Spikes and Glitches: Voltage spikes and glitches are brought about by variation in the supply voltage that may cause processors to skip or misinterpret instructions.

4.4.1 Fault Attack To Recover Symmetric Key

In order to recover the key, we will make use of SEUs to set a bit at a particular position to one. This can be achieved by adjusting the V_{cc} voltage of the circuit appropriately or by using lasers targeted at the particular memory location. For more details regarding bit sets/reset, please refer to section 4.3 in [5]. We are now ready to present our attack which can allow the adversary to recover the error vector $\mathbf{e} = (e_0, e_1)$, which can then be used to compute the symmetric key $K = \mathbf{K}(e_0, e_1)$.

We make the following assumptions about the adversary:

- The attacker has physical access to the circuit that implements the DECAPS($\cdot$) function.
- The attacker only knows the value of the public key $pk = (f_0, f_1)$ and the cipher text c obtained from a previous call to ENCAPS($\cdot$).

Given these assumptions, the adversary $\mathcal{A}$ can recover the symmetric key as follows:

- $\mathcal{A}$ can query the DECAPS($\cdot$) function using the same (pk,c) pair.
- The circuit decodes the ciphertext $\mathbf{c}$ to get the error vector $\mathbf{e}$.
- However, before the circuit can execute the instruction that checks whether the weight of the error vector obtained from decoding is equal to $\mathbf{t}$, we can induce a Single Event Upset(SEU) where we set the first bit of $\mathbf{e}$ to be one.
- Now, when the comparison is done to check the weight of $\mathbf{e}$, a DECAPS *Failure* would imply that the weight of the error vector is no longer equal to t, i.e, we set a bit that was previously set to zero.
- If no decoding error occurs, then the weight did not change and the bit we set to one was already one.
- Thus, we now know the bit present at position 1.
- We can repeat this operation $2r$ times by making $2r$ queries to DECAPS to recover $\mathbf{e}$, where $len(\mathbf{e}) = len(e_0) + len(e_1) = r + r = 2r$.
- Once we have $\mathbf{e}$, we can compute the symmetric key $K(e_0, e_1)$.

Thus, this attack recovers the symmetric key in O(r).

4.4.2 Fault Attacks To Cause Obstruction

Fault attacks can also be used to hinder the execution of the scheme. For instance, an adversary can induce voltage spikes to skip over the instruction in DECAPS that checks whether the error vector has the correct weight, and thus, always guarantee that the receiver computes the symmetric key and never returns a decoding error. This might cause the receiver to compute an incorrect symmetric, especially in the cases where the adversary modifies the ciphertext, when in fact it should be returning a decoding failure. On the other hand, the adversary can also induce random bit flips to cause decoding failures.

The techniques suggested above above correspond to a non-persistent attack, i.e, they do not permanently damage the hardware and the circuit resumes its normal behaviour once the source of the faults is removed. Thus, if the adversary injects these faults at random intervals, it will cause the circuit to behave unpredictably. And since there is nothing inherently wrong with the circuit, the receiver will be unable to identify what causes this erratic behaviour.

5 Modification To Protect Against Our Proposed IND-CCA Attack

From the previous sections, we know that a ciphertext can be represented as $c = (c_0, c_1)$, where $c_i = mf_i + e_i$. We also looked at how an adversary cannot distinguish between a real simulation of the IND-CCA game from a fake simulation in case of decoding failures. Note, here we make the assumption that the hash function $\mathbf{K}$ used to compute the symmetric key is secure, i.e, it is collision and pre-image resistant so that an attacker cannot recover the error vector $\mathbf{e}$ from K and use this $\mathbf{e}$ to carry out a IND-CCA attack, in which case a decoding failure might provide useful information regarding the validity of $\mathbf{e}$ and hence, K.

To protect the scheme against IND-CCA attack, we suggest that the mechanism incorporate the ciphertext in the computation of the symmetric key. The intuition behind this approach is that by making use of the ciphertext to compute K, we restrict the adversary's ability to maul the ciphertext. This approach was inspired by work in [4].

One possible way to make the scheme secure against the attack suggested in this paper is to to compute the symmetric key as follows:

$$K = \mathbf{K}(e_0, m \cdot e_1)$$

where the ENCAPS$(\cdot)$ function chooses m at random. When an adversary modifies the ciphertext $c = (c_0, c_1)$, where, $c_i = m \cdot f_i + e_i$, he is essentially changing the values of either m, f_i and/or e_i. If the adversary modifies the public key component f_i to f_i', then $m \cdot f_i'$ is no longer a valid codeword and hence the syndrome of the ciphertext will not be equal to the syndrome of the error vector. So DECAPS$(\cdot)$ will output a decoding failure.

Similarly, if the attacker modifies the the error vector components e_i's, there are two possible scenarios. In the first scenario, the weight of the modified error is no longer equal to t and so DECAPS$(\cdot)$ outputs a decoding failure. In the second case, the modified error vector also happens to have a weight of t, so decoding succeeds. However, since this error vector is no longer equal to the original error, the symmetric key that DECAPS$(\cdot)$ outputs will be some random value K" which does not provide any information regarding the original error e. Moreover, if the length of the string that $\mathbf{K}$ outputs is x, then the probability that $K" = K^*$ is $\frac{1}{2^x}$, which is fairly low, given that x is chosen carefully. We would also like to note, that throughout this process, we assume that $\mathbf{K}$ is a secure hash function, i.e, it is pre-image and collision resistant. Thus, if the adversary modifies the ciphertext so that DECAPS$(\cdot)$ outputs $K" = K$, then it means that the adversary has found a collision in $\mathbf{K}$ which contradicts our assumption. The pre-image resistant property of $\mathbf{K}$ also ensures that the adversary cannot deduce any useful information regarding the inputs of $\mathbf{K}$ by observing its output. Thus, comparing K" does not provide the adversary with any useful information.

The only other option for the adversary to modify the ciphertext is to change the value of m. Our attack exploited the fact that the value of m, which is chosen randomly during ENCAPS$(\cdot)$ does not affect the decoding and key computation process and hence can be altered to produce a new ciphertext. However, with the modification in the key computation step, changing the value of m changes the input to $\mathbf{K}$, thus resulting in a random K", which with high probability, does not equal either K or K^* following the same arguments as above. Again, the comparison of the observed output of DECAPS$(\cdot)$ does not provide any useful information.

Thus, by implementing these modifications, BIKE-1 can be made secure against the IND-CCA attack that is suggested in this paper. However, as the authors noted in their paper, it is difficult to make BIKE completely IND-CCA secure. This is because, nowadays, the bit flipping decoding techniques do not attain a negligible decoding failure rate. Thus, attacks such as the GJS attack can be used to recover the private key h, which in turn can be used to recover m, e and ultimately K. However, the authors also suggest that there is a possibility of proving that certain decoding techniques exists that have a negligible failure rate. Therefore, if decoding techniques that achieve negligible failure rates can be proven to exist, then for correctly chosen parameter, the above suggested modification may be generalized to provide IND-CCA security.

6 Protection Against Fault Attacks

Depending upon the type of fault attacks, there various different approaches that can be taken to make the system secure. Since the attack described in this paper is primarily a hardware attack, the circuit implementing the scheme can implement some hardware countermeasures. For instance, to mitigate single event upsets brought about by making use of lasers and variation in supply voltage, the circuit can make use of light detector to detect changes in the gradient of light. The circuit can also make use of supply voltage detectors

to detect abrupt changes in the supply voltage and ensure that the voltage is within the circuit's tolerant threshold. This will prevent the adversary from causing skips in the execution of instructions.

7 Conclusion

In this paper, we have performed the security analysis of the BIKE-1 key encapsulation mechanism against chosen plaintext attacks, chosen ciphertext attacks, reaction attacks and fault attack. We further provided an attack that demonstrated that BIKE-1 is not indistinguishable under chosen ciphertext attacks. We also demonstrated a potential fault attack that can be used to recover the symmetric key.

Finally, we suggested a slight modification to the existing scheme so that it makes use of the ciphertext in computation of the symmetric key. This modification makes the scheme indistinguishable under chosen ciphertext attacks if a decoding technique can be proved to have negligible failure rates. Since this does not currently exist, we strongly insist that the scheme remains as defined by the authors.

Overall, BIKE has great potential as a candidate for the post-quantum cryptography standard. As mentioned in [1], having chosen ciphertext security can be challenging, but that does not mean it should not be attempted. We presented a possible modification to further secure BIKE against chosen ciphertext attacks in order to make BIKE an even stronger candidate for the standard of post-quantum cryptography. While this modification is not possible at the moment, it at least shows a bright future for BIKE if a decoding algorithm can be proven to have negligible failure rates.

8 Future Work

While BIKE is a promising candidate of a Key Encapsulation Mechanism, its security has only been tested in the context of a classical random oracle model (ROM), i.e, the hash function $\mathbf{K}$ behaves as a classical ROM. However, it will be interesting to see whether its security holds under a Quantum Random Oracle Model (QROM)[7], where an adversary is allowed to make quantum queries to the oracle. Another possible direction of research is to prove the existence of decoding techniques that achieve a negligible decoding failure rate. If such techniques can be shown to exist, then it will be possible to modify BIKE so that it is IND-CCA secure.

From an application perspective, one possible direction to proceed in would be to perform a thorough code review of the current implementation and identify any security bugs or vulnerabilities that can be used to exploit the system. If the code satisfies the security requirements, we could also look into incorporating this scheme as a cryptographic suite into OpenSSL/SSH.

References

[1] https://bikesuite.org/files/BIKE.pdf, BIKE: Bit Flipping Key Encapsulation

[2] J. Mceliece, R. (1978). A Public-Key Cryptosystem Based on Algebraic Coding Theory. JPL DSN Progress Report. 44.

[3] R. Misoczki, J. Tillich, N. Sendrier and P. S. L. M. Barreto, "MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes," 2013 IEEE International Symposium on Information Theory, Istanbul, 2013, pp. 2069-2073

[4] Jiang H., Zhang Z., Chen L., Wang H., Ma Z. (2018) IND-CCA-Secure Key Encapsulation Mechanism in the Quantum Random Oracle Model, Revisited. In: Shacham H., Boldyreva A. (eds) Advances in Cryptology – CRYPTO 2018. CRYPTO 2018. Lecture Notes in Computer Science, vol 10993. Springer, Cham

[5] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," in Proceedings of the IEEE, vol. 94, no. 2, pp. 370-382, Feb. 2006. doi: 10.1109/JPROC.2005.862424

[6] 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.

[7] Boneh D., Dagdelen O., Fischlin M., Lehmann A., Schaffner C., Zhandry M. (2011) Random Oracles in a Quantum World. In: Lee D.H., Wang X. (eds) Advances in Cryptology – ASIACRYPT 2011. ASIACRYPT 2011. Lecture Notes in Computer Science, vol 7073. Springer, Berlin, Heidelberg