A Closer Look at the Guo–Johansson–Stankovski Attack Against QC-MDPC Codes

Tung Chou, Yohei Maezawa, and Atsuko Miyaji

Graduate School of Engineering, Osaka University
2-1 Yamadaoka, Suita, Osaka 〒565-0871, Japan
blueprint@crypto.tw, maezawa@cy2sec.comm.eng.osaka-u.ac.jp,
miyaji@comm.eng.osaka-u.ac.jp

Abstract. In Asiacrypt 2016, Guo, Johansson, and Stankovski presented a reaction attack against QC-MDPC McEliece. In their attack, by observing the difference in failure rates for various sets Φ_d of error vectors, the attacker obtains the distances between 1's in the secret key and can thus recover the whole secret key. While the attack appears to be powerful, the paper only shows experiment results against the bit-flipping algorithm that uses precomputed thresholds, and the explanation of why the attack works does not seem to be convincing.
In this paper, we give some empirical evidence to show that the Guo–Johansson–Stankovski attack, to some extent, works independently of the way that the thresholds in the bit-flipping algorithm are chosen. Also, by viewing the bit-flipping algorithm as a variant of "statistical decoding", we point out why the explanation of the Guo–Johansson–Stankovski paper is not reasonable, identify some factors that can affect the failure rates, and show how the factors change for different Φ_d.

1 Introduction

In 1978, McEliece presented in his seminal paper [6] the first code-based public-key encryption system. The paper opens the area of code-based cryptography, which is considered as an important branch of the post-quantum cryptography today. The McEliece crpytosystem has stood firmly for 40 years and is thus considered rather confidence-inspiring. However, the public-key size (typically at the scale of 1 megabyte) makes it hard to deploy the scheme in some scenarios.

In order to solve the problem of key size, in 2013, Misoczki, Tillich, Sendrier, and Barreto introduced the usage of QC-MDPC codes for McEliece [7]. Compared to the conventional McEliece system, QC-MDPC enjoys much smaller

This work is partially supported by JSPS KAKENHI Grant (C) (JP15K00183), Microsoft Research Asia, CREST (JPMJCR1404) at Japan Science and Technology Agency, the Japan-Taiwan Collaborative Research Program at Japan Science and Technology Agency, and Project for Establishing a Nationwide Practical Education Network for IT Human Resources Development, Education Network for Practical Information Technologies. Permanent ID of this document: eac422391e669b6d7bbaf8d29c49d2ad. Date: 2018.11.2.

key sizes (typically at the scale of a few kilobytes). Despite the large advantage in key size, the decoding algorithm, the so-called "bit-flipping algorithm", is a probabilistic algorithm. Even worse, there is no satisfying way to evaluate the decoding failure rate when using the bit-flipping algorithm.

In 2016, Guo, Johansson, Stankovski presented in their paper [8] a reaction attack against QC-MDPC McEliece. In their attack, by observing the difference in failure rates for various sets Φ_d of error vectors, the attacker obtains the distances between 1's in the secret key and can thus recover the whole secret key. The Guo–Johansson–Stankovski attack appears to be quite effective as long as the decoding failures can be observed.

To show the effectiveness of the attack, [8] uses one specific variant of the bit-flipping algorithm: the variant that uses precomputed thresholds. As shown in some papers (e.g., [9]), there are many variants which performs better (in terms of decoding failure rate) than the one with precomputed thresholds. This invokes the natural questions: is the Guo–Johansson–Stankovski attack still effective when applied to other variants? Also, although some arguments are given in [8] to show why the attack works, the arguments are unfortunately not convincing to us.

In this paper, we first show that the Guo–Johansson–Stankovski attack works against a rather conservative variant of the bit-flipping algorithm. From the results, we conclude that there might be some factor that naturally causes different failure rates for different Φ_d's. We then discuss about one such factor and show the corresponding experiment results. Furthermore, we also discuss how to view the bit-flipping algorithm as statistical decoding. From such a viewpoint, it can be seen why the explanation in [8] does not seem to be reasonable, and it is shown in detail how the various factors which can affect the failure rates change between different Φ_d.

The organization of the paper is as follows. Section 2 gives a review on some basic concept related to the Guo–Johansson–Stankovski attack. Section 3 discusses about the effectiveness of Guo–Johansson–Stankovski attacks against different variants of the bit-flipping algorithm. Section 4 discusses how the bit-flipping algorithm can be viewed as statistical decoding and identifies two factors that can affect the failure rate. Section 5 tries to give a unified view of how Guo–Johansson–Stankovski attack works in the CPA case and the CCA case.

2 Preliminaries

In this section, we give a brief review on the basic concepts of QC-MDPC codes, the bit-flipping algorithm, and the Guo–Johansson–Stankovski attack.

2.1 QC-MDPC Codes

"MDPC" stands for "moderate-density-parity-check". As the name implies, an MDPC code is a linear code with a "moderate" number of non-zero entries in a parity-check matrix H. In some sense, MDPC codes are simply LDPC codes

with H with sufficiently high density such that H cannot be easily recovered when used in code-based cryptography (which is a rather ambiguous definition).

For ease of discussion, in this paper it is assumed $H \in \mathbb{F}_2^{r\times n}$ where $n = 2r$, even though some parameter sets in [7] allow $n = 3r$ or $n = 4r$. H can be viewed as the concatenation of two square matrices, i.e.,

$$H = \left[H^{(0)}|H^{(1)}\right],$$

where $H^{(i)} \in \mathbb{F}_2^{r\times r}$. Each row of H contains a moderate number of 1's.

"QC" stands for "quasi-cyclic". Being quasi-cyclic means that each $H^{(i)}$ is cyclic. For ease of discussion, one may consider

$$H^{(k)}_{(i+1) \bmod r,(j+1) \bmod r} = H^{(k)}_{i,j},$$

even though [7] allows a row permutation on H. Note that being quasi-cyclic implies that H has a fixed row weight w. The following is a quasi-cyclic matrix with $r = 5$ and $w = 4$:

$$\begin{pmatrix} 1\,0\,1\,0\,0 & 0\,1\,0\,0\,1 \\ 0\,1\,0\,1\,0 & 1\,0\,1\,0\,0 \\ 0\,0\,1\,0\,1 & 0\,1\,0\,1\,0 \\ 1\,0\,0\,1\,0 & 0\,0\,1\,0\,1 \\ 0\,1\,0\,0\,1 & 1\,0\,0\,1\,0 \end{pmatrix}.$$

One can use QC-MPDC codes for the McEliece (as in [7]) or Niederreiter [10] (as in [11]) cryptosystem. One noticeable difference between the QC-MDPC McEliece/Niederreiter and traditional McEliece/Niederreiter is that there is no need to permute the columns to obtain the public keys; the public keys are just systematic generating matrices or systematic parity-check matrices. This allows us to maintain the quasi-cyclic structure and thus save the public-key size.

The number of errors a code is able to correct is often denoted as t. Since there is no good way to figure out the minimum distance for a given QC-MDPC code, t is usually merely an estimated value. In this paper, unless explicitly stated otherwise, we will consider the parameter set $(r, w, t) = (4801, 90, 84)$. This parameter set is evaluated to have a 80-bit security level in [7], and it is the one targeted by [8].

For the discussion in this paper, we reintroduce the concept of "distance spectrum" presented in [8]. The distance spectrum $D(v)$ for $v \in \mathbb{F}_2^r$ is defined as the multi-set that contains distances of 1's in the vector, where the distance is defined in a cyclic way:

$$D(v) = \{\min(j - i,\ i - j + r) \mid v[i] = v[j] = 1,\ i < j\}.$$

The distance spectrum of a $r \times r$ cyclic matrix is defined to be the distance spectrum of any row of the matrix. We use $D(v)[d]$ to denote the multiplicity of $d \in \{1, \ldots, \lfloor r/2 \rfloor\}$ in the distance spectrum.

2.2 The Bit-flipping Algorithm

As described in [7], the bit-flipping algorithm is a probabilistic, iterative algorithm for decoding LDPC codes. The algorithm takes a noisy codeword $y = c+e$ as input. In each iteration, some of the entries of (the current version of) the noisy codeword y' are considered to be more likely to be erroneous, and the bits are flipped to obtain a new (possibly) noisy codeword. In the simplest version of the algorithm, iterations are repeated until a codeword c' is reached. Our hope is that $c = c'$ so that decoding is successful.

Each iteration starts with computing the syndrome s of the current noisy codeword. Each entry s_i then indicates whether the noisy codeword satisfies the corresponding parity-check equation or not: if $s_i = 1$, the noisy codeword does not satisfy the parity check defined by the i-th row of H (denoted by H_i). The number of unsatisfied parity checks

$$u_j = |\{H_{i,j} \mid s_i = 1\}|$$

for each position of the n positions are then collected to form a vector u. The vector u serves as an indicator of how likely it is for the positions to be in error: the larger u_j is, the more likely the position is presumed to be in error. Then, y'_j is flipped if the corresponding u_j is considered to be "large enough". Note that a simple way to compute u is to sum up all H_i such that $s_i = 1$, where the summation is done in $\mathbb{Z}^n$.

Now the remaining problem is, which bits should be flipped given the vector u? In [7] two possibilities are given:

- Flip y'_i if $u_i \geq T_j$, where T_j is a precomputed threshold for the iteration j.
- Flip y'_i if $u_i \geq \max(u) - \delta$, where δ is a predefined small value ([7] proposed to use $\delta = 5$).

We note that, as shown in [9], there are also many other ways to set the thresholds. In particular, [9] proposed to modify y'_i in an "in-place" fashion; one can consider this as allowing to flip at most one y'_i in each iteration. In the remaining of this paper, we will focus on "out-of-place" decoders such as the ones given in [7], where in each iteration we flip all the y'_i with u_i greater or equal to the threshold.

2.3 The Guo–Johansson–Stankovski Attack

In [8], Guo, Johansson, and Stankovski presented an attack against the QC-MDPC McEliece scheme [7]. Their attack is a reaction attack: the attacker sends a bunch of ciphertexts to the private-key holder, and by observing whether the decodings are successful or not, the attacher is able to recover the secret key. They showed that the attack works in two settings, the "CPA case" and the "CCA case".

The CPA case essentially means that the sender is able to choose the error vector for each ciphertext. The attack works as follows.

1. For each distance d in $\{1, \ldots, \lfloor r/2 \rfloor\}$, generate a set Φ_d of weight-t error vectors. Each element $e \in \Phi_d$ has about t/2 pairs on 1's that are separated by distance d, and all the 1's lie in the first block of e.
2. Send the vectors in all Φ_d to the secret-key holder and observe the failure rate P_d for each Φ_d.
3. Generate a figure that shows the points (d, P_d). P_d will then form several non-overlapping groups, and each d will then be classified into one of the group based on P_d. The group with the highest failure rate will then contain all the distances with multiplicity 0, and the group with the second highest failure rate will contain distances with multiplicity 1, and so on.
4. With the multiplicities for each d, which essentially shows $D(H^{(0)})$, run [8, Algorithm 2] to obtain $H^{(0)}$. The algorithm essentially enumerates all candidates of $H^{(0)}$ that fit the distance spectrum. Once $H^{(0)}$ is obtained, $H^{(1)}$ can also be obtained easily.

In the CCA case, the sender does not have the ability to choose the error vector. One can consider that the error vectors are hash outputs. The attack works as follows.

1. Generate a bunch of ciphertext and let Φ be the corresponding set of error vectors. For each distance d in $\{1, \ldots, \lfloor r/2 \rfloor\}$, define Φ_d to be the set that contains all elements in Φ that has distance d; that is,

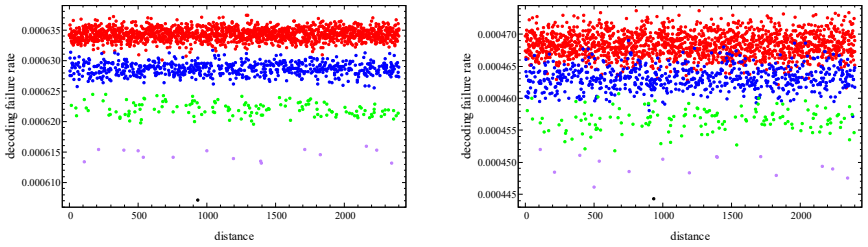
$$\Phi_d = \{e \in \Phi \mid d \in D(e^{(0)})\}$$

2. Send the vectors in all Φ and observe the failure rate P_d for each Φ_d.
3. Generate a figure that shows the points (d, P_d). P_d will then form several non-overlapping groups, and each d will then be classified into one of the group based on P_d. The group with the highest failure rate will then contain all the distances with multiplicity 0, and the group with the second highest failure rate will contain distances with multiplicity 1, and so on.
4. With the multiplicities for each d, which essentially shows $D(H^{(0)})$, run [8, Algorithm 2] to obtain $H^{(0)}$. Once $H^{(0)}$ is obtained, $H^{(1)}$ can also be obtained easily.

Note that, to demonstrate the effectiveness of the attack, [8] uses the bit flipping algorithm with precomputed thresholds (without specifying the actual thresholds). There is no evidence in [8] that the attack can work when the thresholds are chosen in other ways, e.g., when thresholds are set to be $\max(u) - \delta$. In [8] some arguments are given to show why the attack works, but as we will discuss in Section 4.2 the explanation has some flaws.

3 Effectiveness of the Guo–Johansson–Stankovski Attack

In [8], it is shown that the attack works when the thresholds are predefined fixed values. This naturally causes the doubt that whether the attack can really works when the thresholds are chosen in other ways (in particular, in more conservative



(a) CPA case (b) CCA case

Fig. 1. Decoding failure rates for Φ_d's. For the CPA case $(n, r, w, t) = (9602, 4801, 90, 84)$. For the CCA case $(n, r, w, t) = (9602, 4801, 90, 90)$. For the CPA case $|\Phi_d| = 10^6$. For the CCA case we generate is $2.4 \cdot 10^9$ error vectors in total.

ways). In this section, we try to give some empirical evidence and argue that the Guo–Johansson–Stankovski attack, to some extent, is independent of the way the thresholds are chosen.

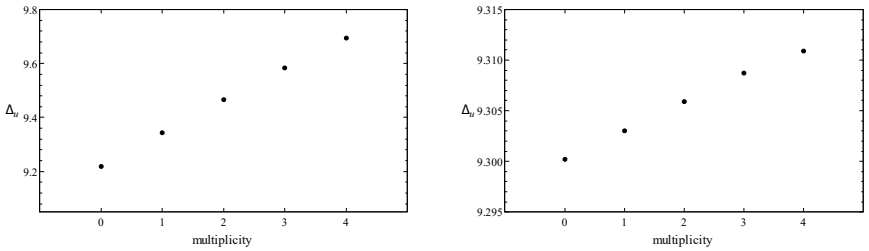
3.1 Experiment Results

To understand how the Guo–Johansson–Stankovski attack behaves when the thresholds are chosen in other ways, we consider the other option described in [7]: to use $\max(u) - \delta$ as the thresholds. In particular, we use $\delta = 0$, as this is more conservative than any $\delta > 0$. Using $\max(u)$ as thresholds is apparently the most conservative strategy among the "out-of-place" decoders. The results are in Figure 1. Note that, for the CCA case, in order to increase the failure rate, we increase t to 90. As shown in the figure, the failure rate decreases as the multiplicity increases. Such phenomena has been shown in [8] for precomputed thresholds.

3.2 An Indicator of the Hardness of Decoding

The experiment result in the previous subsection causes the following questions to rise: can it be that the Guo–Johansson–Stankovski attack actually works independent of the thresholds? In other words, is there some reason that makes it intrinsically easier to decode vectors in Φ_d when the multiplicity of d gets larger? To answer the question, we would like to have an (possibly heuristic) indicator for the hardness of decoding when given in H and e. Our hope is that the indicator shows that Φ_d gets harder to decode as the multiplicity of d gets larger.

Recall that in each iteration of the bit-flipping algorithm, the r_i's with larger u_i's are flipped. As some non-erroneous positions r_i might have u_i that are greater than the threshold, it is possible that some non-erroneous positions are flipped. Roughly speaking, how much the u_i's for the erroneous positions and the



(a) The CPA case (b) The CCA case

Fig. 2. Relationship between Δ_u and multiplicity.

non-erroneous positions are separated from each other determines how likely it is to distinguish the two cases. To quantify the idea, we thus consider the differences of the erroneous positions and the difference of non-erroneous positions. In other words, we define

$$\Delta_u = \sum_{e_i=1} u_i/t - \sum_{e_i=0} u_i/(n-t)$$

and use it as the indicator.

The experiment results are given in Figure 2. As shown in the figure, in both CPA and CCA cases, Δ_u increases as the multiplicity increases. Therefore, when Δ_u is considered, it seems that it should be easier to decode when the multiplicity increases. This matches our experiment results in the previous subsection and the results shown in [8].

4 A Deeper Look

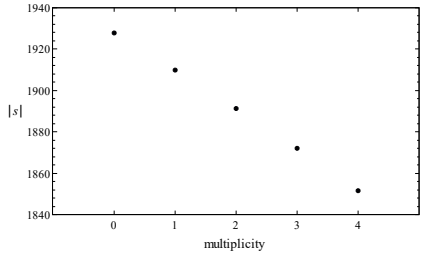
In this section, we take a deeper look at the behaviour of the bit flipping algorithm to see what makes the difference in Δ_u. In particular, we will consider the bit-flipping algorithm in the view of "statistical decoding" and identify the factors that affect Δ_u.

4.1 Statistical Decoding

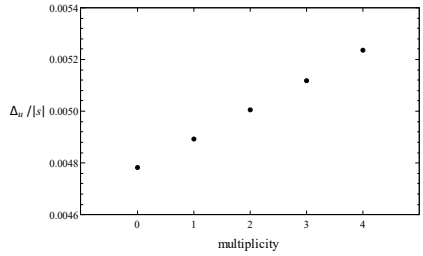
The statistical decoding algorithm described in [12] works as follows. Given a noisy codeword $y = c + e \in \mathbb{F}_2^n$ and a reasonably-large set $H_w \subset \mathbb{F}_2^n$ of weight-w vectors, the algorithm starts with computing

$$u = \sum_{h \in H_w,\ hy^T=1} h \in \mathbb{Z}^n.$$

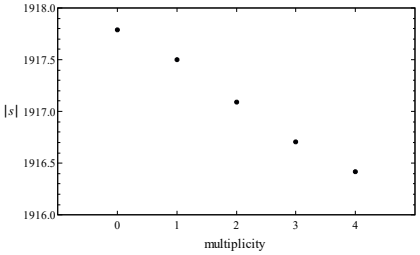
Then a set $I = \{i_1, \ldots, i_k\}$ is chosen such that $u_{i_1}, \ldots, u_{i_k}$ are the smallest entries in u. The set I is then considered as the "information set" (which means



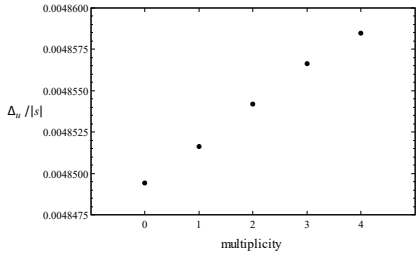
(a) $|s|$ for each multiplicity, CPA case



(b) $\Delta_u/|s|$ for each multiplicity, CPA case



(c) $|s|$ for each multiplicity, CCA case



(d) $\Delta_u/|s|$ for each multiplicity, CCA case

Fig. 3. The values for $|s|$ and $\Delta_u/|s|$ for each multiplicity.

e_{i_j} are all 0), which can be used to decode y easily. Note that finding H_w can be a hard problem itself.

At this moment the reader should notice that statistical decoding is quite similar to the bit-flipping algorithm. Indeed, by letting H_w be the set of rows of the sparse parity-check matrix, the bit-flipping algorithm works essentially in the same way as statistical decoding. Therefore, the two algorithms can be considered to work in the same spirit: the only difference is that the bit-flipping algorithm works in an iterative way, and instead of the smallest entries in u the bit-flipping algorithm consider the largest entries in order to locate the positions in error.

In [12], it was discussed why such a simple algorithm actually works. For each h such that $hy^T = 1$, since the weight of h is only w, the non-zero entries are more likely to be in error. Let i be a nonzero position in h and $hy^T = 1$, let p_w^+ be the probability that $e_i = 1$ and q_w^+ be the probability that $e_i = 0$, [12] pointed out that, as long as $w < n/2$, we have $p_w^+ > q_w^+$. Such bias means that we can obtain a small amount of information about e for each h that satisfies $hy^T = 1$, and summing up all such h's results in the vector u where the erroneous positions tend to have larger values and the non-erroneous positions tend to have smaller values.

We can view such a bias in a different but equivalent way. For each h with $hy^T = 1$, the entries in u which correspond to the non-zero entries in h will be increases. The bias stated above indicates that, on average, the increase in the erroneous positions (u_i with $e_i = 1$) must be larger than the increase in the non-erroneous positions (u_i with $e_i = 0$). To be more precise, let

$$\ell = |\{j \mid h_j = e_j = 1\}|;$$

$\sum_{e_i=1} u_i/t$ would increase by ℓ/t, while $\sum_{e_i=0} u_i/(n-t)$ would increase by $(w-\ell)/(n-t)$. Taking $(n, r, w, t) = (9602, 4801, 90, 84)$ as example, this means that each h would create a difference of at least

$$1/84 - (90-1)/(9602-84) \approx 0.00255.$$

By summing up all such h's, a noticeable difference in $\sum_{e_i=1} u_i/t$ and $\sum_{e_i=0} u_i/(n-t)$ can be created.

From the view above, there are two important factors that would affect the result.

- The first factor is the size of H_w. It is apparently desirable to have a larger H_w so that we can make Δ_u larger. In the bit flipping algorithm, the size of H_w corresponds to the syndrome weight $|s|$, as each row of the parity-check matrix can be considered as h in statistical decoding.
- The second important factor is how much we can separate the two cases for each h, on average for each h with $hr^T = 1$. Note that the larger ℓ is, the more we can separate the two cases. From the view of the bit-flipping algorithm, this is simply $\Delta_u/|s|$.

We thus look at the relationship between $|s|$, $\Delta_u/|s|$ and the multiplicity; the result is shown in Figure 3. As shown in the figure, when the multiplicity increases, $|s|$ decreases while $\Delta_u/|s|$ increases. Since Δ_u also increases with the multiplicity, it is clear that $\Delta_u/|s|$ increases at a faster rate than $|s|$ decreases. From the view of statistical decoding, when considering $|s|$ only, it seems that the failure rate should get higher as the multiplicity increases. However, the increase in $\Delta_u/|s|$ probably compensates for the decrease in $|s|$, and thus eventually we still have a lower failure rate for hight multiplicity. We note that similar observation on $|s|$ has been presented in [13], so we do not take the credit for this part.

The results in Figure 2 and 3 do not depend on the thresholds of the bit-flipping algorithm. However, as the change in the distribution of u for different multiplicity is visible to the decoder, it seems possible to design the thresholds such that the failure rate increases as the multiplicity increases. We thus stress that the discussion above on the easiness of decoding with respect to $|s|$ and $\Delta_u/|s|$ does not take into account how the thresholds are determined.

4.2 Explanation of Guo–Johansson–Stankovski Paper

In [8], in addition to the description of the attack itself, the authors also tried to give some arguments about why the attack works. Similar to the discussion

in the previous subsection, for each row (say row i) of H, we can define

$$\ell_i = |\{j \mid H_{i,j} = e_j = 1\}|.$$

Then ℓ_i indicates how many entries of u will be correctly changed (and thus how many will be wrongly changed), after adding H_i into u. Indeed, assuming that $s_i = 0$, $w - \ell_i$ entries would be correctly changed and ℓ_i would be wrongly changed; Assuming that $s_i = 1$, ℓ_i entries would be correctly changed and $w - \ell_i$ would be wrongly changed. One can thus obtain the following table (essentially [8, Table 2]).

ℓ_i	#(right change)	#(wrong change)
0	w	0
1	1	$w-1$
2	$w-2$	2
3	3	$w-3$
$\vdots$	$\vdots$	$\vdots$

[8] thus concludes that H_i's with an even ℓ_i help to decode, while H_i's with an odd ℓ_i gives a negative effect on decoding. [8] also concludes that it is desirable to have more H_i's with smaller, even ℓ_i.

To show the relationship of the argument above and the attack, they consider two cases in the CPA scenario:

- In CASE-0, the error vectors are from Φ_d where $d \notin D(H^{(0)})$, while
- in CASE-1, the error vectors are from Φ_d where $d \in D(H^{(0)})$.

Their experiment results are shown in the following table (essentially [8, Table 3]).

ℓ_i	CASE-0	CASE-1
0	0.4485	0.4534
1	0.3663	0.3602
≥ 2	0.1852	0.1864

As shown in the table, in CASE-1 the ratio of $\ell_i = 0$ increases and the ratio of $\ell_i = 1$ decreases. It is argued in [8] that both changes are in favor of decoding, and this is why we see a lower failure rate for larger multiplicity. It is not discussed in [8] the impact of $\ell_i \geq 2$.

From the perspective of statistical decoding, the explanation in [8] does not make sense. In particular, from the perspective of statistical decoding, the H_i's with odd ℓ_i's are the ones which help to decode, while those with even ℓ_i's do

Table 1. Relationship between ratios of ℓ_i and the multiplicity.

case	mult.	ℓ_i=0	1	2	3	4	5	6	7	8
CPA	0	0.450502	0.365833	0.141735	0.034886	0.006130	0.000819	0.000087	0.000007	0.000000
	1	0.454535	0.360821	0.140756	0.035992	0.006771	0.000994	0.000118	0.000012	0.000000
	2	0.458623	0.355647	0.139898	0.037070	0.007395	0.001171	0.000152	0.000016	0.000000
	3	0.462740	0.350338	0.139189	0.038161	0.008007	0.001354	0.000187	0.000022	0.000000
	4	0.466780	0.345376	0.138350	0.039124	0.008576	0.001527	0.000240	0.000029	0.000000
CCA	0	0.451770	0.362290	0.141901	0.036170	0.006754	0.000985	0.000117	0.000011	0.000001
	1	0.451843	0.362204	0.141877	0.036192	0.006767	0.000988	0.000118	0.000012	0.000001
	2	0.451934	0.362088	0.141858	0.036218	0.006779	0.000992	0.000118	0.000012	0.000001
	3	0.452020	0.361981	0.141838	0.036241	0.006792	0.000997	0.000119	0.000012	0.000001
	4	0.452088	0.361898	0.141819	0.036260	0.006802	0.001000	0.000120	0.000012	0.000001

not help. As an extreme example, if all $\ell_i \in \{0, 2\}$, then $u = \mathbf{0}$, which does not help to decode.

Nevertheless, we follow the approach in [8] to collect the ratios for all possible values for ℓ_i. The results are given in Table 1. The CASE-0 and CASE-1 in [8, Table 3] corresponds to multiplicity 0 and non-zero multiplicities. Also, in [8, Table 3] all the cases with $\ell \geq 2$ are considered as one case. Therefore Table 1 is much more detailed than [8, Table 3].

We note that it is possible to derive $|s|$ and $\Delta_u/|s|$ from Table 1. Let $T(m, \ell)$ be the entry of Table 1 for multiplicity m and $\ell_i = \ell$ (for one of the CPA and CCA case). Then it is easy to see that, for multiplicity m, $|s|$ is simply

$$(T(m, 1) + T(m, 3) + T(m, 5) + T(m, 7)) \cdot r,$$

while $\Delta_u/|s|$ is simply

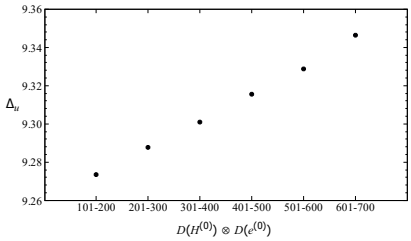
$$\frac{T(m, 1) \cdot \delta(1) + T(m, 3) \cdot \delta(3) + T(m, 5) \cdot \delta(5) + T(m, 7) \cdot \delta(7)}{T(m, 1) + T(m, 3) + T(m, 5) + T(m, 7)},$$

where $\delta(i) = i/t - (w-i)/(n-t)$. It is probably not so easy to see directly how the $|s|$ changes when multiplicity increases. However, as the ratio of H_i with $\ell_i = 1$ decreases and all the ratio of H_i with $\ell_i = 3, 5, 7$ increases (and as $T(m, 1)$ dominates $T(m, 3), T(m, 5), T(m, 7)$), it is clear that $\Delta_u/|s|$ also increases.

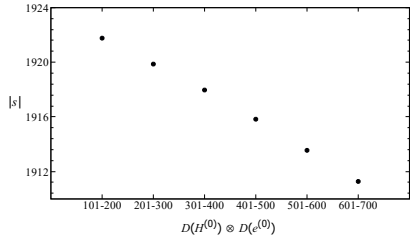
5 A Unified View Between the CPA and CCA Cases

In [8], and also in the experiments for Figure 1, 2, 3 and Table 1, we consider Φ_d with difference definitions in the CPA and the CCA case, but eventually we observe similar changes in the failure rates, Δ_u, $|s|$, and $\Delta_u/|s|$. This makes us wonder if there is a unified way to consider the results for the CPA and the CCA case. In other words, perhaps increasing the multiplicity of d causes some factor to change in a similar way for the CPA and CCA case, and the factor is what is really causing the change in Δ_u, $|s|$, and $\Delta_u/|s|$.

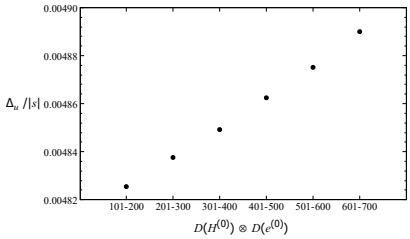
From the experiments for the CPA case, it appears that Δ_u increases, $|s|$ decreases, and $\Delta_u/|s|$ increases roughly linearly as the multiplicity in $D(H^{(0)})$



(a) Relation of Δ_u and $D(H^{(0)})\otimes D(e^{(0)})$



(b) Relation of $|s|$ and $D(H^{(0)})\otimes D(e^{(0)})$



(c) Relation of $\Delta_u/|s|$ and $D(H^{(0)}) \otimes D(e^{(0)})$

Fig. 4. Relationship between Δ_u, $|s|$, $\Delta_u/|s|$ and $D(H^{(0)}) \otimes D(e^{(0)})$

increases. In addition, as the syndrome can be considered as

$$h^{(0)}(x) \cdot e^{(0)}(x) + h^{(1)}(x) \cdot e^{(1)}(x)$$

where $h^{(i)}(x)$ and $e^{(i)}(x)$ are corresponding polynomials of $H^{(i)}$ and $e^{(i)}$ in $\mathbb{F}_2[x]/(x^r+1)$ (as explained in [11]), it seems that the role of $e^{(0)}(x)$ and $h^{(0)}(x)$ are interchangeable. Therefore, it seems reasonable to assume that Δ_u would increase, $|s|$ would decrease, and $\Delta_u/|s|$ would increase roughly linearly as the multiplicity in $D(e^{(0)})$ increases. One evidence that supports this assumption is that we observe similar but much smaller changes in Δ_u, $|s|$, and $\Delta_u/|s|$ in the CCA case compared to the CPA, and in the CCA case the multiplicity of d in some $e \in \Phi_d$ is much smaller than that for the CPA case (at least $\lfloor t/2 \rfloor$).

Based on the discussion above, as increasing the multiplicity of d in $H^{(0)}$ and multiplicity in $e^{(0)}$ should both have help to increase the change in Δ_u, $|s|$, and $\Delta_u/|s|$, it seems reasonable to assume that the factors change linearly with

$$D(H^{(0)}) \otimes D(e^{(0)}) = \sum_i D(H^{(0)})[i] \cdot D(e^{(0)})[i].$$

Based on this assumption, we carried out experiments and present the results in Figure 4. Interestingly, as shown in the figure, as $D(H^{(0)}) \otimes D(e^{(0)})$ increases, similar linear changes in Δ_u, $|s|$, and $\Delta_u/|s|$ can be observed as in Figure 2 and 3.

The experiment results in [13, Fig. 3] might seem a bit similar to our results. We note that $|s|, \Delta_u/|s|$, and Δ_u are all threshold-independent, while numbers of iterations in [13, Fig. 3] are threshold-dependent.

References

[1] Jung Hee Cheon, Tsuyoshi Takagi (editors), *Advances in cryptology—ASIACRYPT 2016, 22nd international conference on the theory and application of cryptology and information security, Hanoi, Vietnam, December 4–8, 2016, Proceedings, Part I*, Lecture Notes in Computer Science, 10031, Springer, 2016. ISBN 978-3-662-53886-9. See [8].

[2] Guido Bertoni, Jean-Sébastien Coron (editors), *Cryptographic hardware and embedded systems—CHES 2013, 15th international workshop, Santa Barbara, CA, USA, August 20–23, 2013, proceedings*, Lecture Notes in Computer Science, 8086, Springer, 2013. ISBN 978-3-642-40348-4. See [9].

[3] Benedikt Gierlichs, Axel Y. Poschmann (editors), *Cryptographic hardware and embedded systems—CHES 2016, 18th international conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, Lecture Notes in Computer Science, 9813, Springer, 2016. ISBN 978-3-662-53139-6. See [11].

[4] Lynn Margaret Batten, Reihaneh Safavi-Naini (editors), *Information security and privacy, 11th Australasian conference, ACISP 2006, Melbourne, Australia, July 3-5, 2006, Proceedings*, Lecture Notes in Computer Science, 4058, Springer. ISBN 3-540-35458-1. See [12].

[5] Tanja Lange, Rainer Steinwandt (editors), *Post-quantum cryptography—9th international conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, Lecture Notes in Computer Science, 10786, Springer, 2018. ISBN 978-3-319-79062-6. See [13].

[6] Robert J. McEliece, *A public-key cryptosystem based on algebraic coding theory*, JPL DSN Progress Report (1978), 114–116. URL: http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF. Citations in this document: §1.

[7] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, Paulo S. L. M. Barreto, *MDPC-McEliece: New McEliece variants from moderate density parity-check codes*, IEEE International Symposium on Information Theory (2013), 2069–2073. URL: http://eprint.iacr.org/2012/409.pdf. Citations in this document: §1, §2.1, §2.1, §2.1, §2.1, §2.2, §2.2, §2.2, §2.2, §2.3, §3.1.

[8] Qian Guo, Thomas Johansson, Paul Stankovski, *A key recovery attack on MDPC with CCA security using decoding errors*, in Asiacrypt 2016 [1] (2016), 789-815. Citations in this document: §1, §1, §1, §1, §2.1, §2.1, §2.3, §4, §4, §2.3, §2.3, §2.3, §3, §3.1, §3.2, §4.2, §4.2, §4.2, §4.2, §4.2, §4.2, §4.2, §4.2, §4.2, §4.2, §4.2, §5.

[9] Stefan Heyse, Ingo von Maurich, Tim Güneysu, *Smaller keys for code-based cryptography: QC-MDPC McEliece implementations on embedded devices*, in CHES 2013 [2] (2013), 273–292. URL: http://eprint.iacr.org/2015/425.pdf. Citations in this document: §1, §2.2, §2.2.

[10] Harald Niederreiter, *Knapsack-type cryptosystems and algebraic coding theory*, Problems of Control and Information Theory **15** (1986), 159–166. Citations in this document: §2.1.

[11] Tung Chou, *QcBits: constant-time small-key code-based cryptography*, in CHES 2016 [3] (2016), 280-300. Citations in this document: §2.1, §5.

[12] Raphael Overbeck, *Statistical decoding revisited*, in ACISP 2006 [4] (2006), 283–294. Citations in this document: §4.1, §4.1, §4.1.

[13] Edward Eaton, Matthieu Lequesne, Alex Parent, Nicolas Sendrier, *QC-MDPC: a timing attack and a CCA2 KEM*, in PQCrpypto 2018 [5] (2018). Citations in this document: §4.1, §5, §5.