DU-QS22: A Dataset for Analyzing QC-MDPC-Based Quantum-Safe Cryptosystems

Mohammad Reza Nosouhi[1,2(✉)], Syed W. Shah[1,2], Lei Pan[1,2], and Robin Doss[1,2]

[1] Centre for Cyber Security Research and Innovation, Deakin University, Geelong, Australia
{m.nosouhi,syed.shah,l.pan,robin.doss}@deakin.edu.au
[2] Cyber Security Cooperative Research Centre (CSCRC), Joondalup, Australia

Abstract. Cryptographically Relevant Quantum Computers (CRQC) will likely compromise the security of current Public-Key Encryption (PKE) mechanisms and make them unusable in the near future. In view of this, the National Institute of Standards and Technology (NIST) is currently undertaking the standardization of post-quantum Key Encapsulation Mechanisms (KEM) such that they can withstand quantum-capable attackers. One potential standardization candidate (i.e., BIKE) is based upon Quasi-Cyclic Moderate Density (QC-MDPC) codes and offers benefits in terms of security and key-size compared with other candidates. Since this candidate is highly dependent upon the performance of the decoder employed in the decapsulation subroutine, we in this paper, present a dataset for benchmarking the performance of various instantiations of decoders that may be proposed by the wider research community in future. To the best of our knowledge, no other dataset exists for researchers to benchmark their decoders for QC-MDPC-based cryptosystems.

Keywords: PKE · KEMs · Quantum-safe · QC-MDPC-based cryptosystem

1 Introduction

KEMs play a critical role in ensuring the security of today's communication systems. However, their reliance on public-key infrastructure (e.g., RSA [1]) will make them vulnerable in the future if CRQCs are ever built. In view of these anticipated problems, NIST is currently undergoing a standardization process to propose alternate KEMs such that they are deployable on contemporary devices and will be quantum-safe – i.e., they will not be compromised if an attacker is leveraging a CRQC (such as nation state actors) to launch the attacks.

In the previous round of the NIST standardization process (i.e., Round 3), various schemes for encryptions were shortlisted either as finalists or alternate candidates. These schemes are largely code-based, lattice-based, or isogeny-based mechanisms proposed for the post-quantum era [2]. Of these candidates, code-based schemes are receiving a lot of interest from the wider research community. One attractive code-based KEM

J. Lin and Q. Tang (Eds.): AC3 2022, LNICST 448, pp. 3–10, 2022.
https://doi.org/10.1007/978-3-031-17081-2_1

included in NIST's Round 3 is referred to as Bit Flipping Key Encapsulation (BIKE [9]) that is based upon the QC-MDPC variant of the original code-based cryptosystem proposed by McEliece in 1978 [3]. This variant is interesting because of its smaller key-sizes compared with other proposal. The security of QC-MDPC-based quantum-safe cryptosystems (e.g., BIKE) depends on the Decoder Failure Rate (DFR) of the decoder employed in decapsulation subroutine of the receiver can facilitate some attacks. For example, prior research has indicated the potential of a particular side-channel attack in compromising private-keys by leveraging the DFR [4].

Keeping in view the importance of decoder and decoder failures in the facilitation of side-channel attacks, the research community has proposed various decoders for code-based cryptosystems. For example, different instantiations of decoders are proposed in [5–7], all based on the Bit-Flipping decoder proposed in [8]. However, the horizon scan of related literature identifies no dataset that can be used to benchmark different instantiations of decoders (i.e., both previously proposed and any future modifications). Specifically, in decoder DFR evaluations, the decoder needs to be tested with a large number of ciphertexts.

Therefore, in this paper, we present a brief overview of our software implementation of QC-MDPC-based cryptosystem (i.e., in MATLAB) and release the resultant dataset that may help in benchmarking the decoders. Precisely, we release randomly generated private keys, the corresponding public keys, and ciphertexts generated using the public keys and random message vectors. The generated keys and ciphertexts have been saved in '*.txt' files for benchmarking purposes. We believe that this dataset will help the wider research community to analyze and benchmark the decoders instead of doing an end-to-end implementation of a cryptosystem that can be both time consuming and inadequate for comparison purposes. In other words, using our dataset, researchers can focus on the decryption (or decoder) subroutine without implementing the key generation and encryption subroutines.

The rest of the paper is organized as follows: Sect. 2 presents a brief overview of the QC-MDPC-based cryptosystem. Section 3 contains an elaboration on the importance of decoders. Sections 4 and 5 contain an overview of our implementation and a description of our dataset, respectively. Finally, our concluding remarks appear in Sect. 6.

2 QC-MDPC-Based Cryptosystem

This section presents a brief overview of the QC-MDPC-based cryptosystem originally proposed by Misoczki et al. in [10]. The block diagram of the original QC-MDPC-based cryptosystem is shown in Fig. 1.

As is evident from Fig. 1, the QC-MDPC-based cryptosystem is comprised of two subroutines – i.e., Encryption and Decryption. However, both subroutines are facilitated by a third subroutine that generates the private and public keys (i.e., Key-Generation Subroutine). In a QC-MDPC-based cryptosystem, a public key is a combination of 'Generator Matrix (**G**)' of the underlying QC-MDPC code and 'a Number (t)' representing the number of errors that need to be inserted in the valid codeword (i.e., the Hamming weight of the error vector **e**). On the other hand, a private key is the 'Parity-Check Matrix (**H**)' of the QC-MDPC code.

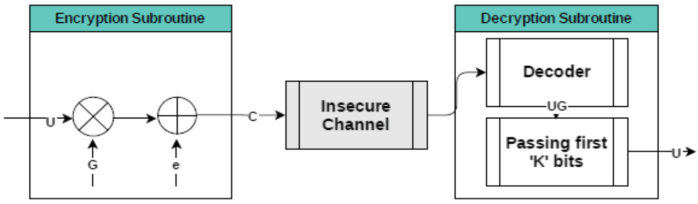

Fig. 1. Block diagram of the original QC-MDPC cryptosystem

In a nutshell, the plaintext '**U**' is multiplied by 'Generator Matrix (**G**)' to get a codeword, in which 't' errors (**e**) are inserted by the sender. This constitutes the ciphertext that is sent to the receiver. The receiver's job is to find the error bits (**e**) and remove them to obtain the valid codeword (**UG**) by leveraging the private key (**H**). It is noteworthy that this constitutes a syndrome decoding problem proven to be NP-complete for a randomly chosen parity check matrix (**H**) [11]. Due to the aforementioned reason, these code-based cryptosystems are envisaged to be quantum-safe.

3 Importance of Decoder

Note that, the decryption process (see in Fig. 1) is mainly reliant upon the decoder (that utilizes the private key). The relevant decoders (such as those proposed in [5–7]) are generally based upon the Bit-Flipping algorithm [8]. The performance of the decoder is highly important in determining the security-level of the QC-MDPC-based cryptosystem (see details in [12]). For example, the GJS attack presented in [4] demonstrated the possibility of a particular side-channel attack on QC-MDPC-based cryptosystems that utilizes the decoder failure rates to fully recover the private key. In addition, a lower decoder failure rate that is needed for the higher security levels generally leads to a larger size of circulant blocks that define the size of private key (see details in [10]). Therefore, in pursuit of low decoder failure rates with relatively smaller key size, researchers continue to propose various instantiations of the Bit-Flipping algorithm.

Typically, the performance of these decoder(s) is analyzed by computing the decoder failures for smaller sizes of circulant blocks and then performing the linear extrapolation to the failures that correspond to the needed level of security (failures of $2^{-\lambda}$ for $\lambda - bit$ security, see further details in [5]). However, no benchmarking dataset exists that can facilitate the analysis of such decoders at the smaller sizes of circulant blocks and their comparison with other proposals for establishing their efficacy in achieving the required level of security. To fill this gap, we perform an end-to-end implementation of the QC-MDPC-based cryptosystem in MATLAB and provide a dataset that will help analyze the decoders and compute the decoder failure rate needed for the target security level (i.e., by performing linear extrapolation).

Next, we present a brief overview of our implementation that facilitates our data generation.

4 Overview of Our Implementation

We implement all three modules of the QC-MDPC-based cryptosystem in MATLAB – i.e., Key-Generation, Encryption, and Decryption modules. The first two modules – i.e., Key-Generation and Encryption facilitate the data generation that are needed for analyzing the decoders and are released as a part of this work. The final module – i.e., the Decryption module is used to present a sample analysis to demonstrate how readers can leverage this dataset. For this demonstration, we present the results obtained through the Black-Grey Flip decoder implemented in MATLAB (the identical decoder is used in BIKE [9]).

4.1 Key-Generation Module

The Key-Generation module generates both public and private keys that are used for codewords generation and decoding, respectively. As pointed out in [13], the generation of public key from a private key needs polynomial inversion that can be computationally laborious for the default method for inversion available in MATLAB. To make this process computationally feasible, we implement the Itoh-Tsuji Algorithm to facilitate efficient inversion through simple cyclic shifts (see details in [14]). In addition, the public-key generation also involves multiplication operations. For the efficient implementation of multiplication, we perform multiplication in a polynomial domain with each coefficient of the product obtained as $c_i = \sum_{j=0}^{r-1} a_j b_{(i-j+r) \ mod \ r} mod 2 \ for \ i = 1, 2, \ldots, r-1$; where '**a**' and '**b**' represents two polynomial and 'r' represents the size of the circulant block in the private key.

4.2 Encryption Module

The Encryption module generates the ciphertexts by utilizing the public-key generated through 'Key-Generation' modules. This is accomplished by multiplying plaintext message '**U**' with '**G**' (again the multiplication operation described above is utilized), that leads to a codeword. The codeword is transformed into ciphertexts by adding random errors '**e**' with weight 't'. The addition is simply done by performing 'XOR' between the codeword and the randomly chosen error vector.

4.3 Decryption Modules

The Decryption module leverages a decoder (i.e., any instantiation of the Bit-Flipping algorithm) that uses a private key (**H**) to eliminate the errors inserted in the codeword by the sender. The Decoder provides the position of bits that are erroneous which are then filliped to obtain the codeword. Once the codeword is obtained, it can be transformed into plaintext by leveraging the systematic form of the public key – i.e., by just passing the first 'K' bits to output ('K' is the size of plaintext).

5 DU-QS22 Dataset

For analyzing the performance of any instantiation of Bit-Flipping decoder, we need ciphertexts and associated private-keys (corresponding to the public-keys) with varying values of sizes of circulant blocks (that define the size of keys, alluded to as 'r' hereafter). In lines with prior works (e.g., [7]), we select the following values of 'r': 9461, 9539, 9643, 9817, and 10009 bits, for dataset generation, with parameters 't' and 'w' both fixed 134 and 142, respectively (- i.e., same as proposed in BIKE, see the details of parameters in [10]). Our analysis suggests that these values of 'r' are sufficient to capture the overall trend of decoder failure and apply linear extrapolation to the point that corresponds to the needed level of security. For each r value, we compute the number of ciphertexts that suffices for obtaining at least 1000 failures before the confidence in decoder failures is established (see the rationale for this threshold in [5]). Note that, the number of generated ciphers is increased since failures are much more difficult to obtain for larger values of 'r' (recall our target is to have 1000 failures as in prior works). Overall, our presented dataset is comprised of public-keys, private-keys, and ciphertext, as listed in Table 1.

Table 1. Size of dataset

'r'	Private-keys	Public-keys	No. of ciphers/key	Total ciphers
9461	20	20	500	10,000
9539	50	50	500	25,000
9643	50	50	1,000	50,000
9817	60	60	5,000	300,000
10009	75	75	10,000	750,000

Our dataset is made publicly available[1,2]. Note that, although only private keys and ciphertexts are needed for analyzing the decoders, we make public-keys publicly available as well (this can help generate additional ciphertexts if needed). All this data is contained in separate *.txt files and can be used to perform analysis irrespective of the platform used for decoding (such as MATLAB that we used for implementing the BGF decoder).

In addition to the keys and ciphertexts shown in Table 1, we also generated 900,000 ciphertexts using 90 keys for $r = 12323$, which is the value of 'r' recommended in the BIKE scheme with the 128-bit security level (see [9] for detail). We also include this data in our repository as these ciphertexts can be used for other purposes such as performance evaluation of decoders (e.g., modifying number of iterations to improve the efficiency) and security evaluations (e.g., analysis of different attack models) at the value of 'r' (i.e., 12323 as reported in BIKE [9]) appropriate for the 128-bit security (i.e., minimum requirement of NIST for standardization).

[1] Dataset is alluded to as DU-QS22 (i.e., Deakin University-Quantum Safe 2022).
[2] Dataset is available https://figshare.com/s/2df4edbbb2eb057e4e30.

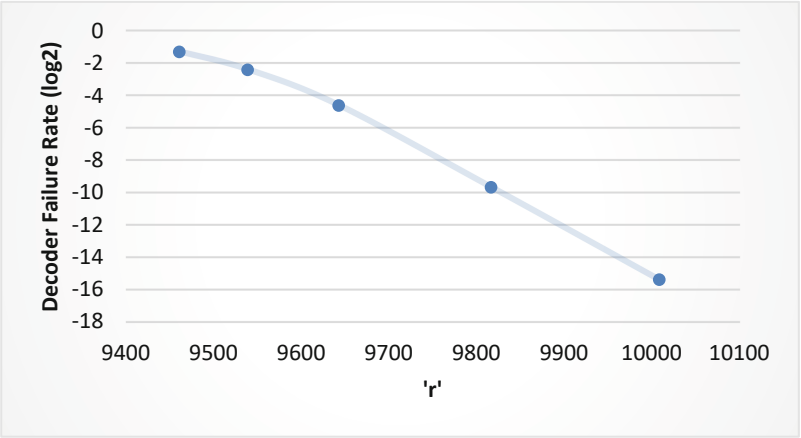

Fig. 2. A sample analysis of a decoder (BGF) with our dataset

How to use these data files? For performing the analysis of decoder, one can simply read the ciphertext files and used the corresponding private keys for decoding. For example, for $r = 10009$, we have 0.75 million ciphers generated through 75 keys. This means that for each key we have 10,000 ciphers. Therefore, for analysis, one can divide the total number of ciphertexts by the number of keys and decode the i_{th} chunk of ciphertexts with i_{th} private key. Note that, for private keys, we only saved the indices of non-zero bits (and thus must be considered accordingly while re-generating the private keys). We have also provided a detailed 'description' with a dataset belonging to different sizes of circulant blocks in our data repository (see the link provided above).

A Sample Analysis with BGF Decoder – To perform a sample analysis with the dataset (described in the previous section), we implemented (in MATLAB) the BGF decoder with the same parameters proposed in BIKE [9]. This decoder is a modification of the Bit-Flipping decoder proposed in [8]. It basically works by transforming a private key into a Tanner graph and computing syndrome vector that helps in determining the number of unsatisfied checks. These unsatisfied checks are then compared with a threshold to find the error bits (see detailed description in [8]). It is noteworthy that, for a constant time implementation the number of iterations is fixed which makes finding the error bits difficult (note that additional masking operations are performed in the BGF decoder – see details in [9]).

For each 'r' value in Table 1, we computed the average decoder failures (with a target of at least 1000 failures) and the resultant curve is shown below in Fig. 2. It is evident from Fig. 2 that the curve is concave (i.e., decaying) thereby making the application of linear extrapolation justified to find the value of 'r' corresponding to a needed level of security (e.g., 128 or 192-bits security that translates to decoder failure rates of 2^{-128} or 2^{-192}, , which is difficult to achieve even on powerful computing platforms, thus extrapolation is the only option). Similarly, one can use this synthetic dataset for analyzing different decoders and the impact of different parameters such as threshold and masking operation as used in the BGF decoder (see details in [9]) and weighting operations (such as those used in [7]).

6 Conclusion

This paper presents a dataset for benchmarking and cross-comparing different instantiation of Bit-Flipping decoders whose optimal performance is crucial for ensuring the needed level of security (e.g., minimum of 128-bits security required for NIST standardization). We also present an overview of our implementation of the QC-MDPC-based cryptosystem in MATLAB and details on how this dataset can be utilized. Finally, we present a sample analysis with the BGF decoder proposed for BIKE using the dataset generated in this work to guide the reader on how to analyze any decoder and apply linear extrapolation on the obtained curve. We maintain that this dataset can help the wider research community in benchmarking and comparing their proposed instantiations of decoders with other proposals and establish their efficacy.

Acknowledgment. This work was partially funded by the Cyber Security Cooperative Research Centre (CSCRC).

References

1. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
2. Beullens, W., et al.: Post-quantum cryptography. Technical report. European Union Agency for Cryptography (2021)
3. McEliece, R.J.: A public-key cryptosystem based on algebraic. Coding Thv. **4244**, 114–116 (1978)
4. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 789–815. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_29
5. Sendrier, N., Vasseur, V.: About low DFR for QC-MDPC decoding. In: Ding, J., Tillich, J.-P. (eds.) PQCrypto 2020. LNCS, vol. 12100, pp. 20–34. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_2
6. Drucker, N., Gueron, S., Kostic, D.: On constant-time QC-MDPC decoding with negligible failure rate. IACR Cryptol. ePrint Arch. **2019**, 1289 (2019)
7. Nilsson, A., Bocharova, I., Kudryashov, B., Johansson, T.: A weighted bit flipping decoder for QC-MDPC-based cryptosystems. In: Proceedings of the 2021 IEEE International Symposium on Information Theory (2021)
8. Gallager, R.: Low-density parity-check codes. IRE Trans. Inf. Theory **8**(1), 21–28 (1962)
9. BIKE. https://bikesuite.org. Accessed 10 Jan 2022
10. Misoczki, R., Tillich, J.-P., Sendrier, N., Barreto, P.S.: MDPC-McEliece: new McEliece variants from moderate density parity-check codes. In: Proceedings of the 2013 IEEE International Symposium on Information Theory, pp. 2069–2073. IEEE (2013)
11. Berlekamp, E., McEliece, R., van Tilborg, H.: On the inherent intractability of certain coding problems. IEEE Trans. Inform. Theory **24**(3), 384–386 (1978)
12. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10677, pp. 341–371. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_12

13. Baldi, M.: QC-LDPC code-based cryptosystems. In: QC-LDPC Code-Based Cryptography. SpringerBriefs in Electrical and Computer Engineering, pp. 91–117. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-02556-8_6
14. Drucker, N., Gueron, S., Kostic, D.: Fast polynomial inversion for post quantum QC-MDPC cryptography. In: Dolev, S., Kolesnikov, V., Lodha, S., Weiss, G. (eds.) CSCML 2020. LNCS, vol. 12161, pp. 110–127. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49785-9_8