Faster Constant-Time Decoder for MDPC Codes and Applications to BIKE KEM

Thales B. Paiva Routo Terada

Institute of Mathematics and Statistics
University of Sao Paulo, Brazil

`tpaiva@ime.usp.br`
`rt@ime.usp.br`

September 19, 2022

BIKE [ABB^{+}21]

- Post-quantum code-based KEM
- Selected by NIST for the 4th round
- Variant of the Niederreiter scheme using quasi-cyclic codes
- Uses an efficient decoder called BGF [DGK20b]
- Better analysis of the Decryption Failure Rate **(DFR)**

  → DFR should be negligible for IND-CCA security

Contributions

- We show some limitations and potential problems with BGF
- We propose a new decoder to solve these problems

  → Lower number of iterations

  → **1.47 speedup** for 256 bits

BIKE

Key Generation

1. $\mathbf{H}_0, \mathbf{H}_1 \leftarrow$ Random binary circulant sparse $r \times r$ matrices
2. Secret key: $\mathbf{H} = [\mathbf{H}_0|\mathbf{H}_1]$ (sparse)
3. Public key: $\mathbf{H}_{\text{Pub}} = \mathbf{H}_1\mathbf{H}_0^{-1}$ (dense)

Encapsulation

1. $\mathbf{e}_0, \mathbf{e}_1 \leftarrow$ Random sparse vectors of r bits and weight $|\mathbf{e}_0| + |\mathbf{e}_1| = t$
2. Return $\mathbf{c} \leftarrow \mathbf{e}_0 + \mathbf{e}_1\mathbf{H}_{\text{Pub}}^\top$

Decapsulation

1. $\mathbf{z} \leftarrow \mathbf{c}\mathbf{H}_0^\top = \mathbf{e}_0\mathbf{H}_0^\top + \mathbf{e}_1\mathbf{H}_1^\top$
   $\mathbf{z}$ is the syndrome of the MDPC code generated by $\mathbf{H}$
2. Use **MDPC decoder** to obtain $\mathbf{e}_0$ and $\mathbf{e}_1$

Decoding $\mathbf{z} = \mathbf{e}_0\mathbf{H}_0^\top + \mathbf{e}_1\mathbf{H}_1^\top$

- $\mathbf{H} = [\mathbf{H}_0|\mathbf{H}_1]$ is sparse
- $\mathbf{e} = [\mathbf{e}_0|\mathbf{e}_1]$ is sparse

  $\Rightarrow$ $\mathbf{z}$ tends to be similar to the columns of $\mathbf{H}$ selected by $\mathbf{e}$
- Similarity index is called UPC: $\texttt{upc}_i = \langle \mathbf{H}_i, \mathbf{z}^\top \rangle \in \mathbb{Z}$
- Decoding is an iterative process of learning $\mathbf{e}$ based on UPC values

BGF: The state-of-the-art BIKE decoder

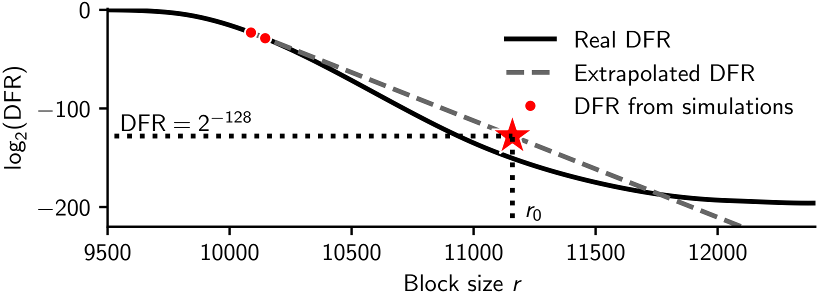
- 5 iterations required for all security levels
- First iteration is expensive but careful to make few mistakes
- Uses a linear function of $|\mathbf{z}|$ to define a UPC threshold

  Remember: high $\texttt{upc}_i \Rightarrow$ high probability that $\mathbf{e}_i = 1$

Achieving negligible DFR [SV20]

- Fix all BIKE parameters except block size r
- **Concavity assumption**

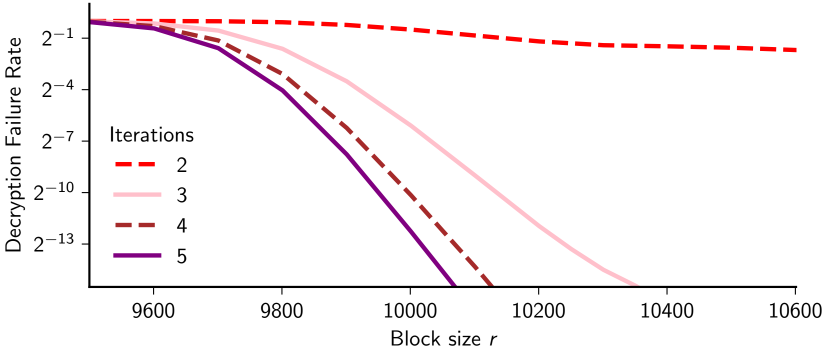
  $\log \mathrm{DFR}(r)$ is concave in the interval where $\mathrm{DFR} > 2^{-128}$
- Use simulations to find DFR points
- Extrapolate to find r_0 achieving negligible DFR



Questioning BGF's concavity

Can we make BGF faster by using < 5 iterations?

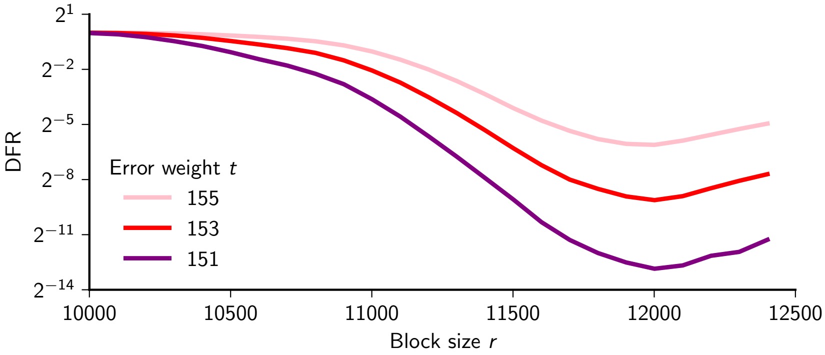
- Problem: for 2 and 3 iterations, BGF's DFR curve is not concave
- Why should we expect that 4 and 5 iterations guarantee concavity?



(128-bit parameters)

BGF with 5 iterations

- Simulations are not enough to find concavity problems
- Consider the DFR for exaggerated error weight $t > 134$



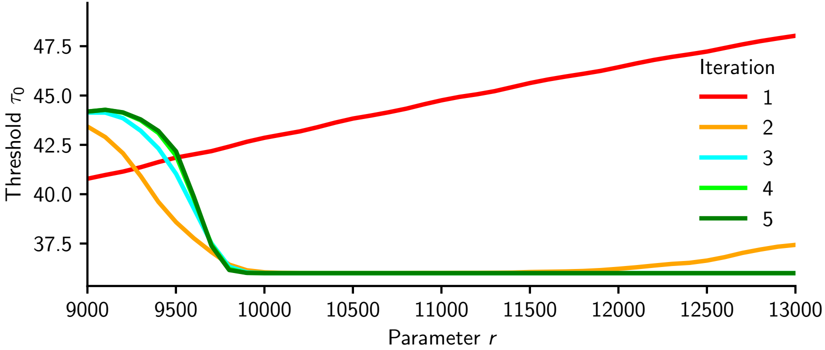
(128-bit parameters: BGF using 5 iterations)

BGF threshold problem

- BGF's problem seems to be in the threshold function

$$\tau_0 = \max\{36, a + b\,|\mathbf{z}|\}$$

- It gets too selective as the block size r grows



(128-bit parameters: BGF using 5 iterations)

Our solution

New decoder called PickyFix uses two procedures

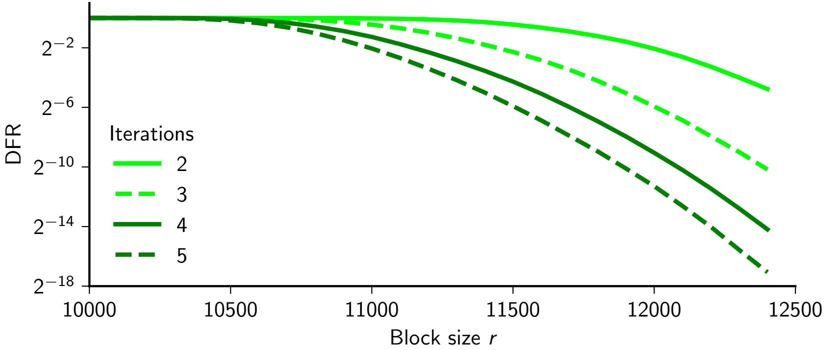
FixFlip → Flips a fixed number n_{Flips} of bits with largest UPCs

- A good value for n_{Flips} is computed by simulations
- It is used in the first iteration to avoid the threshold problem

PickyFlip → Uses different thresholds for flipping

- τ_{Out} to flip 1 to 0
- τ_{In} to flip 0 to 1
- $\tau_{\mathsf{In}} \geq \tau_{\mathsf{Out}}$

PickyFix's concavity for exaggerated errors



(128-bit parameters with $t = 160$)

Implementation

- PickyFlip is easy to implement using BIKE's code [DGK20a]
  The only difference is the threshold selection
- FixFlip's constant-time implementation **is not trivial**
  We describe an efficient procedure to select the highest UPCs
  It is ∼30% slower than PickyFlip

Security level	Iterations	$r = \|\text{public key}\|$	Portable	AVX512
128	2	13,829	1.21	1.18
	3	13,109	1.07	1.08
192	2	27,397	1.31	1.29
	3	25,867	1.14	1.15
256	2	41,411	1.45	1.47
	3	39,901	1.23	1.29

Future Work

Open questions

- Can we use FixFlip for efficient decoding without fixed thresholds?
- Can our implementation be used for more complex thresholds?
- How to strengthen the concavity assumption?
- Is it possible to patch BGF?

Source code and data available at

- `www.ime.usp.br/~tpaiva`
- `https://github.com/thalespaiva/pickyfix`

References I

[ABB^{+}21] Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, Carlos Aguilar-Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor, *BIKE: Bit flipping key encapsulation*, 2021, `https://bikesuite.org/files/v4.2/BIKE_Spec.2021.09.29.1.pdf`.

[DGK20a] Nir Drucker, Shay Gueron, and Dusan Kostic, *BIKE Additional Implementation*, 2020, `https://bikesuite.org/files/round2/add-impl/BIKE_Additional.2020.02.09.zip`.

[DGK20b] Nir Drucker, Shay Gueron, and Dusan Kostic, *QC-MDPC decoders with several shades of gray*, International Conference on Post-Quantum Cryptography, Springer, 2020, pp. 35–50.

[SV20] Nicolas Sendrier and Valentin Vasseur, *About low DFR for QC-MDPC decoding*, PQCrypto 2020-Post-Quantum Cryptography 11th International Conference, vol. 12100, Springer, 2020, pp. 20–34.