Analysedesalgorithmesdedécodagedes codes QC-MDPC

Les codes QC-MDPC permettent la conception du cryptosystème de McEliece avec des clés et une sécurité qui réduit à l'évidence les problèmes de décodage pour les codes quasi-cycliques. En particulier, ces codes sont parmi les plus prometteurs qui sont proposés à l'appel du NIST pour la normalisation de la cryptographie post-quantique.

La première génération d'algorithme de décodage souffre d'un petit, mais pas négligeable, taux d'échec du décodage (DFR, pour *Decoding Failure Rate,* de l'ordre de 10^{-7} à 10^{-10}). Ce qui a permis l'attaque de récupération de clés de Guo, Johansson et Stankovski qui exploite une petite corrélation entre les messages erronés et la clé secrète du cryptosystème de McEliece [42]. Cependant, cette attaque semble ne pas avoir d'incidence sur l'établissement interactif de communications sécurisées (le protocole TLS par exemple), mais l'utilisation de clés publiques statiques pour des applications asynchrones (les courriers électroniques par exemple) est rendue dangereuse.

Il est donc intéressant de comprendre et d'améliorer le décodage des codes QC-MDPC pour les applications cryptographiques. En particulier, la recherche de paramètres pour lesquels le DFR est manifestement négligeable (généralement aussi bas que 2^{-64} à 2^{-128}) augmenterait l'applicabilité du cryptosystème [43].

4.1. Optimisation de l'algorithme *Bit-flipping*

L'algorithme *Bit-flipping* a été décrit dans le chapitre 2. Cependant, il est nécessaire de rappeler de manière brève le fonctionnement de l'algorithme. Ainsi, pour un mot de code $x \in \mathbb{F}_2^n$, on a les étapes suivantes :

- Calculer le syndrome $s = Hx^T$ du mot reçu x.
- Comptez les équations de contrôle de parité insatisfaites, notées $\#_{upc}$ associées à chaque bit de x.
- Inversez les bits de x qui violent plus de b équations, où b est le seuil d'inversion de bit.
- Recalculez le syndrome pour le nouveau x mis à jour.

Ce processus est répété jusqu'à ce que le syndrome devienne nul ou qu'un nombre maximum prédéfini d'itérations soit atteint, à partir duquel une erreur de décodage est renvoyée.

Le nombre d'équations de contrôle de parité insatisfaites est égal au nombre de bits partagés dans une ligne de la matrice de contrôle de parité H et du syndrome s. A noté que le syndrome ne dépend, par définition, que de l'erreur e qui est ajoutée à un mot de code c :

$$S = Hx^T = H(c + e)^T = Hc^T + He^T = He^T$$

Puisque $Hc^T = 0$, par définition.

Ainsi, plusieurs techniques ont été élaborés pour accélérer le calcul du syndrome et réduire le taux d'échec du décodage.

Les décodeurs *Bit-flipping* dans la littérature recalculent le syndrome après chaque itération pour décider si le décodage a réussi ou non. Le coût d'un seul calcul du syndrome peut être estimé à environ deux fois le coût d'un encodage dans le cas des codes QC-MDPC avec $n_0 = 2$.

Les auteurs de [44] proposent une optimisation qui peut être appliquée à tous les décodeurs bit-flipping, basée sur l'observation suivante : si le nombre d'équations de contrôle de parité non satisfaites dépasse un seuil qu'on notera b, le bit correspondant dans le texte chiffré est inversé et le syndrome change. Soulignons que le syndrome ne change pas arbitrairement, mais le nouveau syndrome est égal à l'ancien syndrome accumulé avec la ligne h_j de la matrice de contrôle de parité correspondant au bit inversé à la position j.

En gardant la trace des bits qui sont retournés et en mettant à jour le syndrome en conséquence, le recalcule du syndrome peut être omis. Comme seuls quelques bits sont inversés à chaque itération lors du décodage, la mise à jour du syndrome nécessite beaucoup moins d'ajouts qu'un calcul de syndrome ordinaire.

Il y a deux façons d'appliquer le calcul du syndrome pour les optimisations du paragraphe précédent. L'une d'elles consiste à stocker toutes les modifications du syndrome dans un registre séparé et d'ajouter les modifications à la fin d'une itération de décodage au syndrome. Ainsi, le calcul du syndrome est accéléré mais le comportement du décodage reste inchangé. L'autre possibilité est d'appliquer directement les changements au syndrome à chaque fois qu'un bit du texte chiffré est retourné. Cela accélère également le calcul du syndrome mais affecte également le comportement du décodage puisque le syndrome modifié est utilisé pour déterminer les équations de contrôle de parité non satisfaites de bits de texte chiffré suivants.

4.2. Les variantes de l'algorithme *Bit-flipping*

Les algorithmes itératifs bit-flipping ont un taux d'échec de décodage beaucoup plus élevé lorsqu'ils sont utilisés pour décoder les codes MDPC que ceux des équivalents LDPC. Ceci est dû au fait que la matrice de contrôle de parité d'un code LDPC est sensiblement plus clairsemée qu'une matrice correspondante pour le code MDPC. Autrement dit chaque ligne de la matrice pour le code MDPC contient un plus grand nombre d'entrées non nulles et donc le nombre de bits inclus dans le calcul de chaque bit du syndrome est considérablement augmenté. Raison pour laquelle il est plus difficile de décider si un bit est erroné ou non. Dans la construction originale de Gallager, une seule décision était prise par bit en une itération, et en inversant un bit qui n'est pas en erreur, la probabilité d'une défaillance de décodage est plus élevée.

4.2.1. La variante *Black-Gray*

La variante *Black-Gray* [45] de l'algorithme *Bit-flipping* utilise plusieurs étapes dans une itération de décodage unique afin de réduire la probabilité de retourner les bits inutiles et réduire le taux d'échec. Il utilise deux seuils prédéfinis δ et d pour trier les bits avec un nombre élevé d'équations de contrôles de parité insatisfaisants en deux ensembles : l'ensemble noir et l'ensemble gris.

L'algorithme 10 illustre le décodeur *Black-Gray*. Cette variante fonctionne selon un nombre fixe d'itérations (X_{BG}). L'algorithme comporte trois principales étapes. L'étape I consiste à effectuer une itération de l'algorithme *Bit-flipping* et marquer les bits inversés avec des étiquettes B pour *Black* (noire) et G pour *Gray* (grise). L'étape II consiste à déverrouiller les bits d'erreur noirs qui atteignent un certain seuil. La dernière étape, l'étape III, consiste aussi à déverrouiller les bits d'erreur gris qui atteignent un certain seuil.

Algorithme 10*: Black-gray*

Entrées : Matrice de contrôle de parité H, mot de code c et le maximum d'itérations X_{BG}.

Sorties : L'erreur e.

Exception : ''échec du décodage'' **retourner** $\perp$.

Procedure Black-Gray (c, H)

$s = Hc^T,\ \ e = 0,\ \ \delta = 4$

$B = \phi,\ \ G = \phi$

Pour $itr = (O, \dots, X_{BG} - 1)$ **Faire**

$th = CalculeSeuil(s)$

$upc[n-1:0] = CalculeUPC(s, H)$ // étape I

Pour $(i = 0, \dots, n-1)$ **Faire**

Si $upc[i] \geq th$ **Alors**

$e[i] = e[i] \oplus 1$

$B = B \cup i$ // mise à jour des ensembles noirs

Sinon Si $upc_i >= th - \delta$ **Alors**

$G = G \cup i$ // mise à jour des ensembles gris

$s = H(c^T + e^T)$ //mise à jour du syndrome

$upc[n-1:0] = CalculeUPC(s, H)$ // étape II

Pour $b \in B$ **Faire**

Si $upc[b] > (\frac{d+1}{2})$ **Alors**

$e[b] = e[b] \oplus 1$

$s = H(c^T + e^T)$ //mise à jour du syndrome

$upc[n-1:0] = CalculeUPC(s, H)$ // étape III

Pour $g \in G$ **Faire**

Si $upc[g] > (\frac{d+1}{2})$ **Alors**

$e[g] = e[g] \oplus 1$

$s = H(c^T + e^T)$ //mise à jour du syndrome

Si $(wt(s) \neq 0)$ **Alors**

Retourner $\perp$

Sinon

Retourner e

L'algorithme fonctionne comme suit : les bits avec un nombre maximum d'équations de contrôles de parité non satisfaites sont classés comme des bits *Black* (noirs) et retournés immédiatement. Les bits dont le nombre d'équations de contrôles parité non satisfaits s'écartant du maximum d'un nombre inférieur au seuil δ, sont classées comme des bits *Gray* (gris), mais ne sont pas retournées. Après la boucle initiale, le syndrome et le nombre d'équations de contrôles de parité non satisfaits sont recalculés pour chaque bit. Ensuite, chaque bit dans les deux ensembles de *Black* et de *Gray* sont analysés à nouveau : si le nombre d'équations de contrôles de parité non satisfaites dépasse un seuil défini par le deuxième paramètre d, les bits noirs qui dépassent ce seuil sont ramenés à leur état initial et les bits gris sont retournés. Après chaque étape, le nombre d'équations de contrôles de parité non satisfaisants est mis à jour en conséquence [46].

4.2.2. La variante *Back-flipping*

L'algorithme *Back-flipping* est une variante de l'algorithme originale de Gallager. Cet algorithme fut introduit dans [47] dans le but de réduire le DFR du décodage *Bit-flipping*. Dans ce qui suit, il est décrit comme dans [48].

Comme les positions avec des compteurs plus élevés dans l'algorithme original de Gallager ont des probabilités d'être erronées. Ces positions sont retournées lorsque le compteur est au-dessus d'un seuil, la quantité au-dessus n'a pas d'importance et une partie des informations fiables est perdu. L'idée de *Back-flipping* est d'utiliser ces informations tout en conservant la simplicité du décodeur de *Bit-flipping*.

Parmi les décisions de retournement de bits, la plupart sont bonnes (suppression d'erreurs) et certaines sont mauvais (ajout d'erreurs). Les mauvaises décisions conduisent ainsi à un échec au décodage. Pour exploiter les informations de fiabilité, un décodeur pourrait réduire l'impact des décisions les moins fiables et renforcer l'impact des décisions les plus fiables. *Back-flipping* est un nouvel algorithme de retournement de bits qui utilise le temps τ pour exploiter les informations de fiabilité fournies par les compteurs à chaque retournement de bits. Chaque flip (retournement) a une durée de vie (fini). Lorsque son temps est écoulé, le flip est annulé. Les positions avec un compteur plus élevé restent le plus longtemps retournées que les positions avec un compteur juste au-dessus du seuil. La conception de *Back-flipping* est basée sur les principes suivants :

- Les décisions les plus fiables auront plus d'influence dans le processus de décodage,
- Toutes les mauvaises décisions seront annulées à un moment donné,
- La sélection conservatrice des seuils empêche les mauvaises décisions d'entrer en cascade.

En outre, on constate aisément que, par rapport à l'algorithme *Bit-flipping*, l'algorithme *Back-flipping* ne nécessite que quelques opérations supplémentaires pour gérer un tableau des délais D. De plus, en ce qui concerne le seuil, le ttl ou time-to-live (durée de vie) est très bien approximé par une fonction affine pour tout ensemble de paramètres fixé et son calcul a un coût négligeable en pratique [48].

Algorithme 11*: Back-flipping (*ou *Back-flip)*

Entrées : La matrice de contrôle de parité H, le syndrome s et un entier $u \geq 0$.

Sorties : L'erreur e.

Prérequis : $|s - eH^T| \leq u \quad ou \quad \tau >$ max $\_\tau$.

$e \longleftarrow 0\ ;\ \tau \leftarrow 1\ ;\ D \leftarrow 0$

Tant que $|s - eH^T| > u \quad et \quad \tau \leq$ max $\_\tau$ **faire**

Pour j telque $D_j = \tau$ **faire**

$e_j \leftarrow 0$

$\tau \longleftarrow \tau + 1$

$s' \leftarrow s - eH^T$

$T \leftarrow seuil(|s'|,\ t - |e|)$

Pour $j \in \{0, \dots, n-1\}$ **faire**

Si $|s' \cap h_j| \geq T$ **alors**

$e_j \leftarrow 1 - e_j\ ;\quad D_j \leftarrow \tau + ttl(|s' \cap h_j| - T)$

Retourner e

Règle de sélection des seuils $(\boldsymbol{S, t_0})$. Comme la durée de vie d'un *flip* est toujours fini, un mauvais *flip* sera toujours annulé par la suite. Cependant, il est nécessaire pour éviter d'ajouter d'autres mauvais *flips* pendant la période où il est retourné. Pour y parvenir, les seuils sont utilisés avec $s = |s'|$ et $t' = t - |e|$. Il s'agit de la meilleure estimation du poids d'erreur, qui suppose que chaque flip a supprimé une erreur. Lorsque de nombreuses erreurs ont été ajoutées,

le seuil correspondant est plus élevé que pour l'algorithme original de Gallager, cela ralentira le décodage en laissant le temps d'annuler les mauvaises décisions tout en ne prenant que de nouveaux *flips*. Dans le cas typique, ou la plupart des décisions de *flip* étaient bonnes, le seuil est proche à l'optimal, et le décodage converge rapidement.

Time-to-live $ttl(\delta)$. Empiriquement, il apparaît que la durée de vie devrait augmenter avec la différence δ entre le compteur de position et l'itération seuil. Il devrait également être limité, car sinon des valeurs de comptage aberrantes pourraient conduire à ajouter des erreurs plus difficiles à détecter : des positions correctes avec un compteur élevé peuvent devenir des erreurs avec un compteur bas une fois retournés, leur compteur doit changer radicalement avant d'être corrigées par un algorithme reposant uniquement sur un seuil. La fonction ttl dépend des paramètres du code (en particulier w et t) ainsi que du nombre maximum d'itérations du décodeur.

4.3. Notre proposition

Les variantes *Black-Gray* et *Back-flipping* peuvent décoder efficacement les codes MDPC pour un nombre d'itérations faible (11 itérations au maximum). Or, plus le nombre d'itérations est faible, plus la probabilité de réussir une attaque de réaction (attaque de GJS) est négligeable. Aussi, il a été démontré dans [45] que la réduction du nombre d'itérations permettrait une implémentation en temps constant. Notons qu'une implémentation en temps constant permet d'éviter les attaques temporelles sur le cryptosystème. En effet, ces types d'attaques exploitent la différence du temps d'exécution qui existe entre deux itérations successives pour récupérer la clé secrète. Cependant, si le *Back-flipping* permet d'obtenir un DFR plus bas que le *Black-Gray*, ce dernier est plus performant lorsqu'il s'agit d'une implémentation en temps constant [45].

Ainsi, notre proposition consiste à améliorer le décodage de l'algorithme *Black-Gray* (puisqu'étant plus performant lors d'une implémentation en temps constant) pour obtenir un DFR plus bas ou avoisinant celui de l'algorithme *Back-flipping*.

Le *Black-Gray* comprend trois étapes principales pour chaque itération. Donc plus le nombre d'itérations est grand plus on a d'étapes dans la phase de décodage. L'idée ici est de réduire le nombre d'étapes de l'algorithme et voir comment cela agit sur le DFR. Nous avons ainsi réalisé une combinaison entre le *Black-Gray* et le *Bit-flipping*. C'est-à-dire lors du décodage, nous avons effectué la première itération avec le *Black-Gray* et les itérations restantes avec le *Bit-*

flipping. Nous appellerons cette variante le *Black-Gray-flipping*. La combinaison citée précédemment nous a permis de réduire significativement le nombre d'étapes du décodage, puisque le *Bit-flipping* ne s'effectue qu'en une seule étape pour chaque itération. Par exemple pour cinq itérations, on aura $3 + 1 + 1 + 1 + 1 = 7$ étapes, alors qu'en utilisant le *Black-Gray* seul, on aura $3 + 3 + 3 + 3 + 3 = 15$ étapes. La réduction du nombre d'étapes effectuées par l'algorithme de décodage augmente donc la vitesse d'exécution mais également les performances du décodeur. Reste à savoir comment cela va impacter sur le DFR ?

Pour cela nous avons réalisé des simulations pour comparer le DFR entre le *Black-Gray*, le *Back-flipping* et le *Black-Gray-flipping* pour 11, 7 et 5 itérations.

4.4. Simulations et résultats

4.4.1. Choix des paramètres de simulations

Notre objectif ici est de simuler les trois algorithmes cités précédemment afin de comparer leur DFR. Nous avons ainsi réalisé 2000 essais sur 100 codes QC-MDPC différents et cela pour 11, 7 et 5 itérations. Les paramètres utilisés pour ces codes QC-MDPC sont les suivants :

$$k = r = 4801, \quad n_0 = 2, \quad n = n_0 \times r = 2 \times 4801 = 9602, \quad w = 90 \quad et \quad t = 84$$

Dans ce cas, un bloc de texte en claire de 4801 bits est encodé en un mot de code de 9602 bits, auquel il faut ajouter un certain nombre d'erreurs. La matrice de contrôle de parité se compose ainsi de deux blocs 4801×4801 circulants respectivement pour H_0 et H_1.

Pour la variante *Black-Gray*, les deux paramètres à prendre en compte sont d et δ. Dans [45], les auteurs proposent d'utiliser $\delta = 4$ pour une meilleure optimisation de l'algorithme. Pour le paramètre d, nous avons pris plusieurs valeurs entre 10 et 100 et les simulations montrent que la valeur $d = 60$ donne le DFR le plus bas.

Pour la variante *Back-flipping*, nous considérons les mêmes paramètres que précédemment pour les codes QC-MDPC. Il faudra également fixer un temps maximum de fonctionnement de l'algorithme. Lorsque ce temps est atteint, l'algorithme s'arrête et renvoi un certain nombre d'erreur. Nous avons appelé ce temps maximum de fonctionnement $max\ \_\tau$ et nous l'avons fixé à $max\ \_\tau = 5$ [48].

4.4.2. Simulations avec 11 itérations

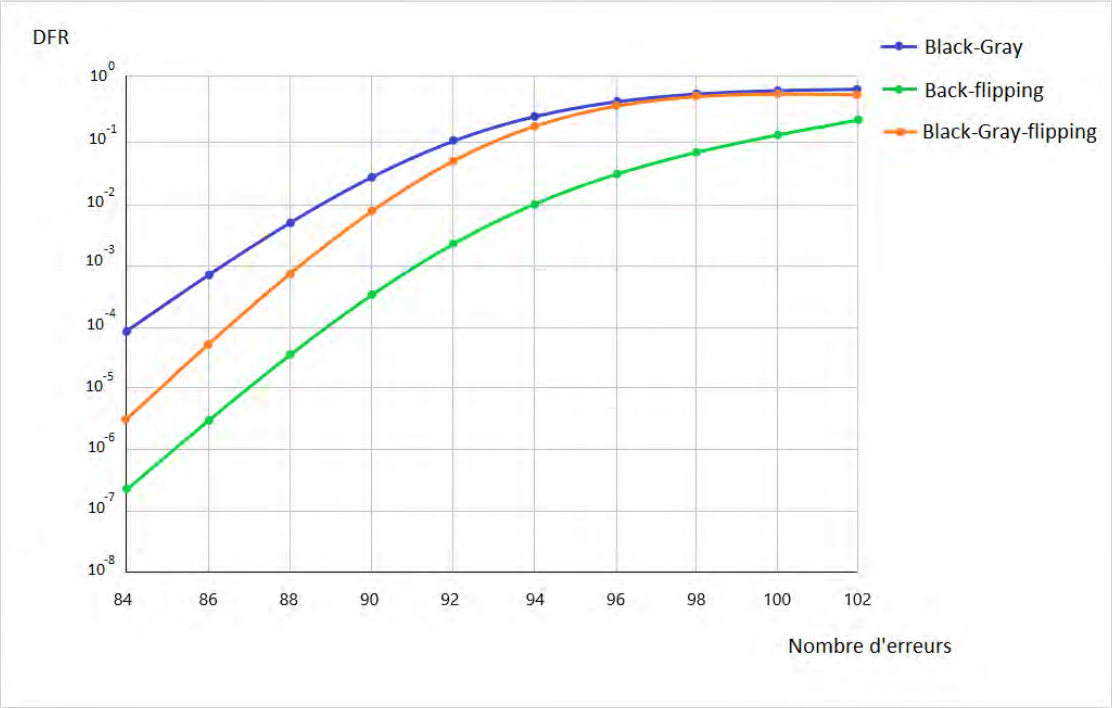

Figure 6 *: Comparaison du DFR des algorithmes Black-Gray, Black-Gray-flipping et Back-flipping pour 11 itérations*

Les simulations pour les algorithmes *Black-Gray*, *Back-flipping* et *Black-Gray-flipping* pour 11 itérations donnent les résultats consignés sur la figure 6 ci-dessus. Ces résultats montrent que l'algorithme *Back-flipping* offre le DFR le plus bas. Cependant, le *Black-Gray-flipping* dépasse de loin le *Black-Gray* original en terme de DFR.

Ceci peut être expliqué par la réduction du nombre d'étapes du décodage pour l'algorithme *Black-Gray-flipping*, mais également par le nombre réduit de bits erronés qui sont retournés. En effet, pour ce dernier, comme le *Black-Gray* divise les bits en erreurs en deux groupes *Black* et *Gray* lors de la première itération, seuls les bits en erreurs se rapprochant du seuil δ sont retournés par le *Bit-flipping* lors des itérations suivantes.

L'algorithme *Back-flipping* utilise le facteur temps pour retourner les bits d'erreurs. Ainsi, comme chaque *flip* a une durée de vie fini, donc un mauvais *flip* est par la suite annulé. Ceci permet d'obtenir un DFR assez bas comparé aux autres algorithmes.

4.4.3. Simulations avec 7 itérations

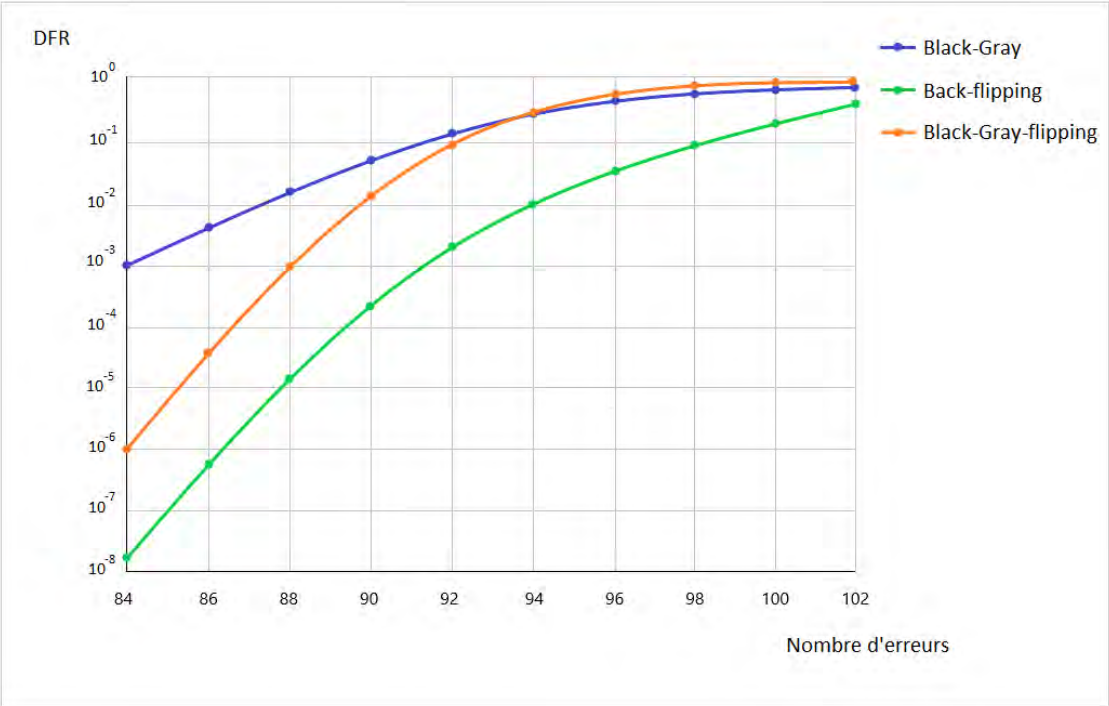

Figure 7 *: Comparaison du DFR des algorithmes Black-Gray, Black-Gray-flipping et Back-flipping pour 7 itérations*

Lorsque l'on réduit le nombre d'itérations à 7, le *Black-Gray* voit son DFR augmenter mais reste toujours performant. En effet, le nombre de bits d'erreurs retournés par l'algorithme est réduit significativement lorsque les itérations sont à leur tour réduites. Ce qui fait que beaucoup d'erreurs ne sont pas corrigées.

Le DFR pour le *Black-Gray-flipping* et le *Back-flipping* diminue avec le nombre d'itérations. Cependant, cette diminution est plus importante pour le *Back-flipping*. En effet, ce dernier est conçu pour fonctionner efficacement avec un minimum d'itérations.

4.4.4. Simulations avec 5 itérations

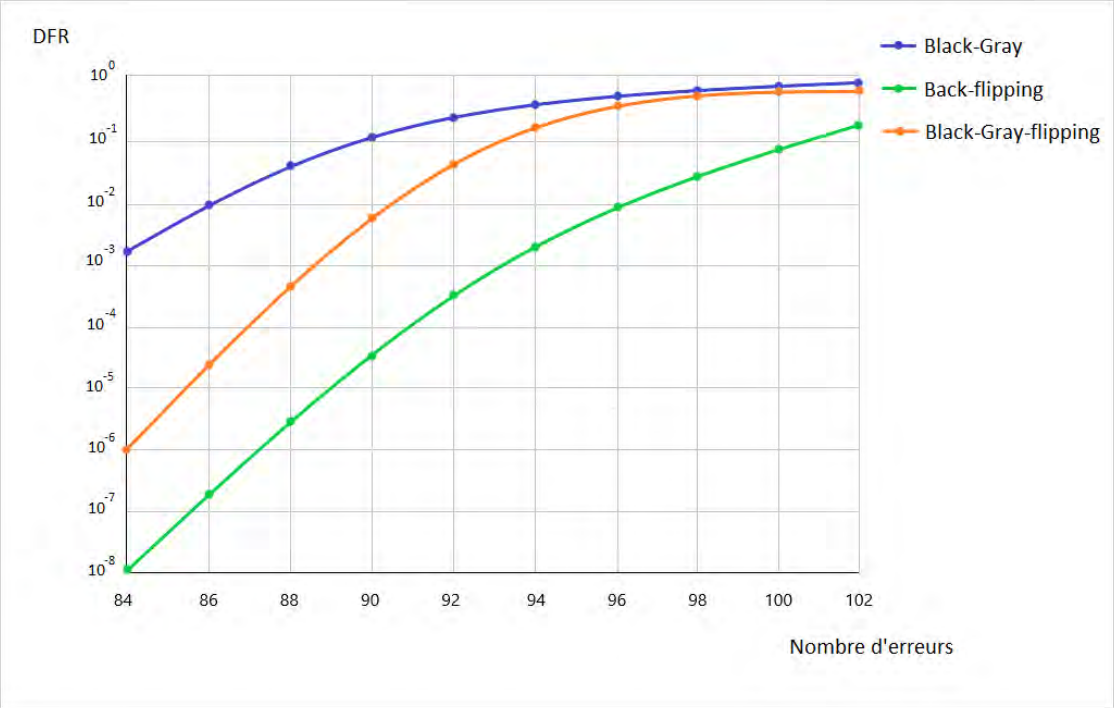

Figure 8 *: Comparaison du DFR des algorithmes Black-Gray, Black-Gray-flipping et Back-flipping pour 5 itérations*

La réduction du nombre d'itérations à 5 fournit des résultats presque similaires que lorsqu'on était à 7 itérations pour les algorithmes *Black-Gray* et le *Black-Gray-flipping*. Les 4 dernières itérations du *Black-Gray-flipping* réalisées à partir du *Bit-flipping* sont à l'origine de cette stabilité du DFR, car le *Bit-flipping* n'étant pas efficace lorsque les itérations sont très réduites.

Le *Back-flipping* est toujours plus performant que les autres algorithmes en terme de DFR. Nous noterons ici une très légère diminution du DFR qui va atteindre 10^{-8}.

L'utilisation de l'algorithme *Bit-flipping* pour le décodage des codes QC-MDPC n'est possible qu'avec un nombre élevé d'itérations pour ainsi parvenir à retrouver efficacement le mot de code. Cependant, plus le nombre d'itérations est grand, plus la probabilité qu'un échec au décodage se produit. Ce qui entraine un DFR assez grand, exploitable pour mener une attaque de réaction (comme celle de GJS) ou une attaque temporelle. Les algorithmes *Black-Gray* et *Back-flipping* sont ainsi conçu pour fonctionner avec un nombre réduit d'itérations. Ils

permettent de décoder efficacement les codes QC-MDPC avec un minimum d'itérations, même si le DFR qu'ils fournissent reste toujours non négligeable.

Cependant, s'agissant d'une implémentation en temps constant, le *Black-Gray* reste plus performant que le *Back-flipping*, même si ce dernier fournit un DFR plus bas. Ce qui nous a poussé à modifier l'algorithme *Black-Gray*. La combinaison entre le *Black-Gray* et le *Bit-flipping* nous a donné l'algorithme *Black-Gray-flipping*.

Nous avons ainsi analysé le DFR pour les différents algorithmes. On note que même si le *Black-Gray-flipping* n'égale pas le DFR du *Back-flipping*, il surpasse de loin le *Black-Gray* original. Maintenant, il reste à étudier les performances de cet algorithme (le *Black-Gray-flipping*) lorsqu'il est implémenté en temps constant.