

# **FCSC 2020**

Merry & Pippin

**Danhia**

danhia@protonmail.com

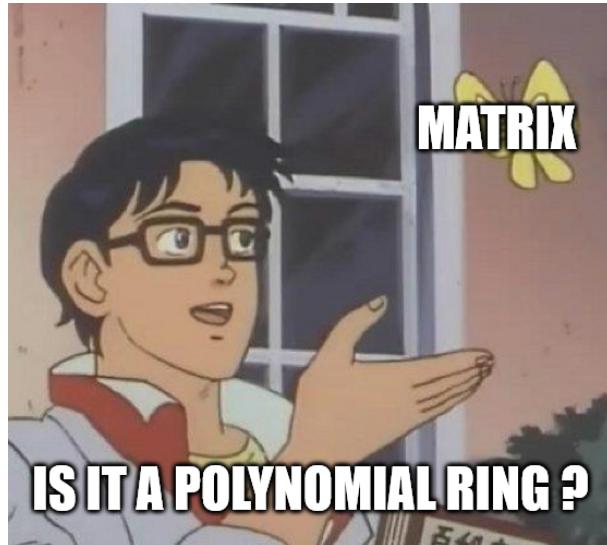
Un Write Up de cryptographie illustré, où l'on ne parle pas de réseaux euclidiens, où l'on n'explique pas comment transformer des polynômes en matrices, mais où l'on débat de l'importance de savoir faire la différence entre les 0 et les -1, et où l'on apprend que le bruteforce résout tous les problèmes mais qu'il vaut mieux optimiser aujourd'hui pour ne pas pleurer demain.

## Merry

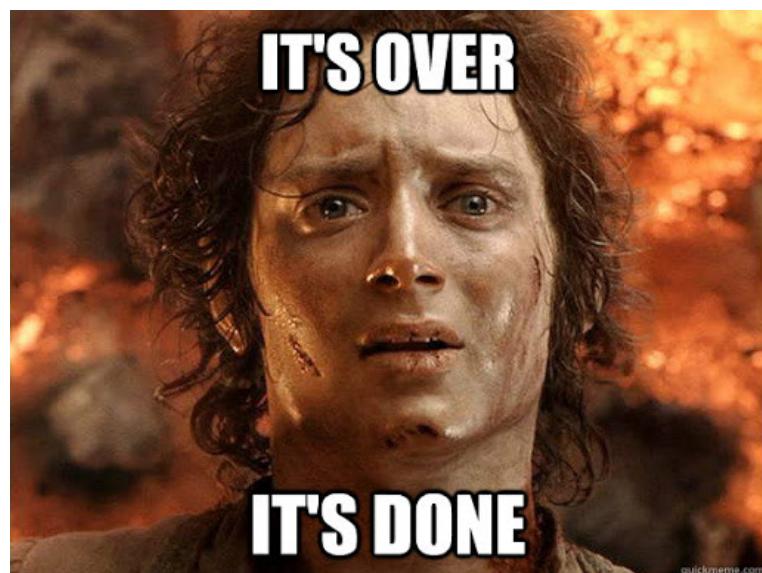
Le code source du serveur est fourni avec le challenge. Il s'agit d'après l'énoncé d'un protocole d'échanges de clés. Contrairement au challenge Deterministic ECDSA, le protocole cryptographique utilisé ne saute pas aux yeux. On se demande s'il s'agit d'un protocole inventé, mais la différence de format entre les clés publiques générées par le serveur (des matrices  $n \times n$  et  $n \times \bar{n}$ ) et les paramètres attendus de la part de l'utilisateur (des matrices  $\bar{m} \times n$  et  $\bar{m} \times \bar{n}$ ) laisse entendre que les opérations réalisées coté client et serveur pour l'échange de clé ne sont pas identiques et qu'on va avoir du mal à déduire le code client du code serveur. Il faut donc partir à la recherche du protocole utilisé qui semble être le Diffie-Hellman du futur.



Le challenge est tagué "post-quantum", donc on commence par consulter un article d'une célèbre encyclopédie en ligne sur le sujet de la cryptographie post-quantique, dans l'espoir que parmi la liste des protocoles l'un corresponde à notre petit bout de code. Les critères identifiés sont pour l'instant : le protocole doit prévoir un échange de clés asymétriques, les clés doivent être représentées par des matrices pour les calculs, et surtout les clés doivent être TRES grandes (la taille de  $A$  est en effet de  $280 \times 280 \times 2^{11}$  = plusieurs centaines de kilo octets). On note deux candidats potentiels : "Ring learning with errors key exchange" ainsi que "Supersingular isogeny key exchange". Cependant notre code source n'a pas grand choses avoir avec les courbes elliptiques et n'est pas non plus basé sur des polynômes (bien qu'on réalise nos calculs dans une structure d'anneau). Retour à la case départ.



En désespoir de cause on se résout à taper dans notre moteur de recherche préféré "post quantum diffie helmann", et après quelques errances on tombe sur un article de Microsoft qui nous renvoie vers le projet d'Open Quantum Safe, qui lui même comporte un lien vers un paper qui va nous accorder l'illumination : <https://openquantumsafe.org/papers/SAC-SteMos16.pdf>. On y découvre en effet l'existence du protocole FrodoKEM, ce qui nous éclaire finalement sur le titre du challenge (Merry de Merry et Pippin, les hobbits du seigneur des anneaux) et nous permet d'achever là notre quête éperdue.



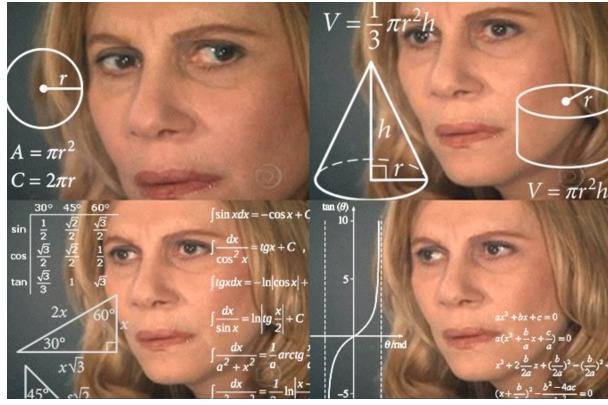
Il est maintenant temps de lire la documentation complète de FrodoKEM et de comprendre enfin ce qu'est censé faire le client (nous) pour pouvoir échanger une clé avec le serveur, et peut-être qu'après on pourra écrire notre première ligne de code plusieurs heures après avoir démarré ce challenge. Vous pouvez trouver ici la spécification complète du protocole : <https://frodkem.org/files/FrodoKEM-specification-20171130.pdf>. Parallèlement à cette lecture, on se met à la recherche d'attaques préexistantes. Malheureusement, FrodoKEM est un protocole assez récent (2017) et il n'y a donc pas beaucoup de littérature à son sujet. On trouve dans une description du protocole les taille minimum recommandée pour ses paramètres :

<b>Scheme</b>	<b><math>n</math></b>	<b><math>q</math></b>	<b>dist.</b>	<b><math>B \cdot \bar{m}^2 = B \cdot \bar{n}^2</math></b>	<b>failure</b>	<b>bandwidth</b>
Challenge	352	$2^{11}$	$D_1$	$1 \cdot 8^2 = 64$	$2^{-41.8}$	7.75 KB
Classical	592	$2^{12}$	$D_2$	$2 \cdot 8^2 = 128$	$2^{-36.2}$	14.22 KB
Recommended	752	$2^{15}$	$D_3$	$4 \cdot 8^2 = 256$	$2^{-38.9}$	22.57 KB
Paranoid	864	$2^{15}$	$D_4$	$4 \cdot 8^2 = 256$	$2^{-33.8}$	25.93 KB

D'après les auteurs, la sécurité du protocole n'est pas garantie pour des paramètres dont la taille serait inférieure à celle de la catégorie "Classical", or nous avons  $n = 280$ ,  $q = 2^{11}$  et  $\bar{m} = \bar{n} = 4$ , soit en dessous de la catégorie "Challenge". Donc on pourrait tenter une attaque sur le problème mathématique sous-jacent qui semblerait être assimilable à celui du SVP (shortest vector problem), à l'aide d'un algorithme dénommé "BKZ". Notre maîtrise des réseaux euclidiens étant ce qu'elle est, on décide de partir sur une autre piste.



Par ailleurs cette attaque ne semblait pas exploiter une des données de l'énoncé : la même bi-clé est utilisée pour plusieurs requêtes ce qui nous fait donc penser à une attaque de "key reuse". On découvre quelques papers traitant de ces attaques sur l'échange de clé RLWE, néanmoins elles s'appliquent au protocole NewHope qui se base sur les polynômes alors que nous sommes toujours sur des matrices. On essaye faiblement de transformer nos matrices en polynômes (ou des polynômes en matrices, à ce stade c'est un peu flou) mais sans succès.



On finit tout de même par comprendre que cela ne servait à rien de comprendre comment l'échange de clés se faisait côté client ni même de songer à l'implémenter, puisque l'attaque la plus simple consiste à envoyer en boucle au serveur des clés fabriquées de toute pièce pour qu'il nous indique lesquelles sont correctes. On va en effet se servir du serveur comme d'un oracle, en espérant que "plusieurs requêtes" signifie en réalité requêtes illimitées. Pour cela pas besoin de réseaux euclidiens, seulement d'un peu de calcul matriciel. On a

$$Key = Dec(C - U \cdot S)$$

avec  $Dec$  une fonction qu'on peut résumer ainsi :  $Dec(x) = 1$  si  $256 < x < 1024$ ,  $Dec(x) = -1$  si  $-1024 < x < -256$ , et  $Dec(x) = 0$  si  $-256 \geq x \geq 256$ . On ne traitera pas les autres cas puisqu'on s'arrangera par la suite pour que  $x$  soit toujours dans un de ces trois intervalles. On va essayer de deviner  $S$  ligne par ligne. En effet, une ligne est de longueur 4 et il y a trois valeurs possibles pour chaque coefficient (0, 1 et -1), donc il faut au maximum  $3^4 = 81$  essais par lignes c'est à dire  $81 \times 280 = 22680$  appels à l'oracle en tout. C'est beaucoup, mais on espère que ça va marcher.

On pose tous les coefficients de  $C$  à 0, ainsi que tous les coefficients de  $U$  sauf  $U[0][0]$  que l'on met à une valeur  $u$  de telle sorte que  $Dec(u) = 1$ . Ainsi

quand le serveur va calculer la clé, on aura

$$Key = \begin{pmatrix} -1 \cdot S[0][0] & -1 \cdot S[0][1] & -1 \cdot S[0][2] & -1 \cdot S[0][3] \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

On peut donc envoyer au serveur nos 81 clés possibles jusqu'à ce qu'il nous indique que la clé est correcte, ce qui nous donnera la première ligne de  $S$ . Pour obtenir les lignes suivantes, il suffit de décaler notre terme non nul  $u$  vers la gauche. Ainsi pour obtenir  $S[i]$  nous devons placer notre  $u$  à la position  $U[0][i]$ . Il suffit de répéter l'opération pour les 280 lignes de  $S$ . Il ne reste alors plus qu'à calculer

$$E = B - A \cdot S \mod q$$

sans oublier de remplacer les 2047 par des  $-1$  et nous pouvons envoyer la paire de clés  $(S, E)$  au serveur, qui nous donne alors le flag.



## Pippin

Ce challenge caché se débloque seulement une fois que Merry a été validé. Parce que Merry et Pippin bien sur. Bon. Nous n'en avons donc définitivement pas fini avec FrodoKEM, on espère juste que ce ne sera pas une trilogie à la seigneur des anneaux. On réouvre la documentation qu'on venait de fermer et on s'y remet.



L'énoncé nous explique que la faille de sécurité permettant l'attaque précédente a été résolue, et que désormais les bi-clés ne seraient réutilisées que pendant un nombre limité de requêtes. On se connecte au serveur et on constate effectivement une limite de 3000 requêtes. L'énoncé nous informe également que certaines valeurs ne sont pas correctement générées, mais nous n'avons pas accès au code source. Un dilemme se pose alors : doit-on tenter de deviner quelles sont les valeurs affectées et monter une nouvelle attaque pour les exploiter, ou bien peut-on essayer d'optimiser notre attaque précédente pour descendre en dessous des 3000 requêtes ?



On envisage brièvement l'hypothèse que  $S$  et  $E$  seraient générées de manière identique, ce qui nous donnerait l'égalité suivante :

$$\begin{aligned} B &= A \cdot S + E \\ &= A \cdot S + S \\ &= (A + I) \cdot S \end{aligned} \tag{1}$$

et on pourrait donc retrouver  $S$  de manière linéaire par une simple inversion de  $A + I$ . Néanmoins après quelques tentatives laborieuses on réalise que  $A$  étant générée de manière aléatoire il y a de fortes chances pour qu'elle ne soit pas inversible. On décide donc d'optimiser l'attaque précédente, et on essaye de trouver une combinaison matricielle qui nous permette de deviner un coefficient en trois tentatives, plutôt qu'une ligne de 4 coefficients en 81 tentatives, ce qui nous ferait un total de  $3 \times 1120 = 3360$  requêtes dans le pire des cas, donc statistiquement plutôt autour de 2240, ce qui serait en dessous de notre limite de 3000 requêtes. On a toujours

$$Key = Dec(C - U \cdot S)$$

On pose tous les coefficients de  $C$  à 0, à l'exception de  $C[0][0]$  que l'on met à une valeur  $c$  et on pose tous les coefficients de  $U$  à 0, à l'exception de  $U[0][0]$  que l'on met à une valeur  $u$ . Ainsi quand le serveur va calculer la clé, on aura

$$Key = Dec \begin{pmatrix} c - u \cdot S[0][0] & -u \cdot S[0][1] & -u \cdot S[0][2] & -u \cdot S[0][3] \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Ainsi, si on veut isoler un seul bit, disons  $S[0][0]$ , il faut choisir  $c$  et  $u$  de telle sorte que tous les coefficients de  $Key$  soit nuls à l'exception de  $Key[0][0]$  et que  $Key[0][0]$  dépende toujours de  $S[0][0]$ . Pour ça il faut choisir  $u$  de telle sorte que  $Dec(u) = 0$ , et  $c$  de telle sorte  $Dec(c - u) \neq 0$ . On généralise aux autres coefficients de  $S$  en décalant  $u$  et  $v$  vers la droite. Malheureusement on constate que ce modèle nous permet seulement de distinguer les coefficients à 0 des coefficients à 1 dans  $S$ , et que les coefficients à -1 seront arrondis à 0 pendant notre attaque. Néanmoins cela nous permet en  $2 \times 1026 = 2052$  requêtes au maximum et 1539 requêtes en moyenne de placer tous les 1 dans notre matrice  $S$ . Il nous reste donc environ 1500 tentatives pour distinguer les 0 des -1.



Pour cela on va reprendre notre première attaque qui consiste à deviner des lignes complètes, sauf qu'ici nous n'avons que 2 possibilités pour chaque coefficient (-1 ou 0) ce qui porte le nombre de lignes possibles à  $2^4 = 16$  pour des lignes comprenant uniquement des 0 à ce stade. En effet, certaines lignes comprennent déjà des 1 qui ne seront pas modifiés par l'attaque, donc pour chaque ligne le nombre de requêtes maximum à effectuer est de  $2^{4-k}$  avec  $k$  le nombre de 1 dans la ligne. Si nous n'avions trouvé aucun 1 à l'étape précédente, nous devrions effectuer  $16 \times 280 = 4480$  requêtes au maximum soit 2380 requêtes en moyenne pour distinguer les -1 des 0. C'est supérieur aux 1500 requêtes qu'il nous reste à utiliser mais si on suppose qu'il y a un 1 par ligne et qu'il nous reste donc seulement trois 0 à tester, on doit effectuer 1120 requêtes en moyenne.

Suite à de savants calculs mathématiques au doigt mouillé que nous ne reproduirons pas ici pour des raisons évidentes de crédibilité, nous décidons qu'il est possible que notre attaque se termine en moins de 3000 requêtes avec une probabilité non négligeable. On lance notre attaque sur le serveur de Merry pour pouvoir tester sa validité sans la contrainte de nombre de requêtes, et on a le plaisir de récupérer une seconde fois le flag au bout de seulement 2638 requêtes, et on se dit que c'était bien plus rapide que la dernière fois et qu'on aurait peut-être pu optimiser notre attaque dès le début. On lance ensuite une première attaque sur le serveur de Pippin, qui dépasse les 3000 requêtes et échoue. On la relance, et elle aboutit finalement au bout de 2297 requêtes et nous délivre le flag.

