Retour d'expérience sur la montée en compétence d'un cabinet d'audit en cryptographie post-quantique

Antoine Gicquel, Alexandre Brenner et Benjamin Sepe
antoine.gicquel@synacktiv.com
alexandre.brenner@synacktiv.com
benjamin.sepe@synacktiv.com

Synacktiv

Résumé. En tant que membre du consortium Hyperform [1] dont l'objectif est de développer des solutions capables de protéger les données sensibles des éventuelles attaques provenant/utilisant des ordinateurs quantiques, Synacktiv est amené évaluer la sécurité des différents démonstrateurs produits par les autres partenaires du consortium. La place de Synacktiv au sein de ce consortium fait suite à une volonté interne de se former sur le sujet, auquel l'entreprise doit se confronter dans d'autres contextes tels que les Certifications de Sécurité de Premier Niveau. Un travail d'auto-formation, complété par une conférence a été suivi par un petit groupe de consultants motivés en interne. Cet article cherche à détailler le processus de montée en compétence de l'équipe, après deux parties de remise en contexte. Il s'attaquera enfin à l'idée reçue selon laquelle la cryptographie post-quantique est un domaine *trop complexe* pour un non-cryptographe, en vulgarisant des algorithmes *post-quantiques* et des attaques sur ceux-ci.

1 Pourquoi s'intéresser à la cryptographie post-quantique ?

La cryptographie moderne se fonde sur des problèmes mathématiques faciles à calculer dans un sens, mais difficiles à inverser. Par exemple :

— L'algorithmes RSA fonde sa robustesse sur le problème de la décomposition d'un grand nombre en facteurs premiers ;

— Les algorithmes de signature sur les courbes elliptiques, ainsi que d'échange de clés via des dérivés du protocole de Diffie-Hellman fondent leur robustesse sur des variantes du problème du logarithme discret.

[1] Hyperform est un projet financé par BPI France dans le cadre de l'appel à projet "Cryptographie Post-Quantique", https://www.bpifrance.fr/nos-appels-a-projets-concours/appel-a-projets-cryptographie-post-quantique

— L'algorithme AES mélange l'entrée et la clé via une suite de permutations, substitutions et combinaisons induisant un nombre astronomique de possibilités rendant le problème insolvable en pratique.

Ces problèmes sont effectivement difficiles à résoudre sur les ordinateurs classiques, que nous connaissons tous. Cependant, le domaine académique s'intéresse depuis de nombreuses années à développer des ordinateurs d'un nouveau genre, manipulant l'information non pas au travers de signaux électriques mais d'états de particules (photons, électrons, ...), répondants aux lois de la physique quantique. Ces ordinateurs, appelés "ordinateurs quantiques", raisonnent de manière probabiliste ; lors du déroulement d'un algorithme, toutes les valeurs de sortie sont initialement possibles. Une suite d'opérations sur l'état interne permet de renforcer la probabilité associée au résultat attendu, tandis qu'elle affaiblit la probabilité d'obtenir tous les autres résultats. À l'issue de l'algorithme, une mesure de l'état interne de l'ordinateur est effectuée, renvoyant un seul des états possibles.[2] La mesure correspond au résultat attendu de l'algorithme, avec une très forte probabilité. Cette nouvelle approche du calcul modifie notre perception de difficulté des problèmes mathématiques sous-jacents à la cryptographie moderne, d'où l'intérêt académique initial de confronter la cryptographie à ces nouveaux ordinateurs.

Les ordinateurs quantiques requièrent des conditions opérationnelles exceptionnelles, notamment concernant la température de fonctionnement, la stabilité des particules utilisés et la fiabilité des appareils de mesure de l'état de ceux-ci pour obtenir des résultats cohérents. De fait, les modèles actuels développés par Google, IBM, Amazon et d'autres géants de la tech sont toujours au stade de preuves de concept, et n'ont pour l'instant pas démontré leur capacité à réaliser des opérations complexes qui ne soient pas déjà solvables à l'aide d'ordinateurs classiques.

Parmi les opérations et algorithmes qu'il devrait être possible d'effectuer à l'aide d'un ordinateur quantique, deux algorithmes retiennent particulièrement l'attention des cryptographes : les algorithmes de Shor et de Grover. En effet, ceux-ci mettent à mal le logarithme discret et la factorisation pour Shor, la recherche exhaustive pour Grover, des problèmes perçus comme "difficiles" pour les ordinateurs actuels et sur lesquels repose la cryptographie d'aujourd'hui. Bien qu'une connaissance fine du fonctionnement interne de ces algorithmes ne soit pas nécessaire pour aborder la cryptographie post-quantique, nous proposons d'en décrire les grandes lignes dans la partie suivante, afin de mieux saisir leurs implications sur

[2] Les physiciens quantiques parlent de "réduction du paquet d'onde" [27]

la cryptographie actuelle. La section suivante peut donc être passée par un lecteur pressé sans entraver la compréhension du reste du papier.

1.1 Les algorithmes quantiques

L'algorithme de Grover L'algorithme de Grover est un algorithme permettant la recherche exhaustive d'un élément dans un ensemble à N éléments, en une complexité temporelle de $\sqrt{N}$ (et une complexité spatiale de $log(N)$) En d'autres termes, cet algorithme permettrait par exemple de trouver la clé AES128 (parmi les 2^{128} clés possibles) permettant de déchiffrer correctement un message chiffré en 2^{64} opérations ($= \sqrt(2^{128})$), au lieu des 2^{128} actuellement nécessaires à un ordinateur classique via une attaque par force brute.

Il a été démontré [28] que cet algorithme atteint la borne inférieure en terme de complexité temporelle sur le problème de la recherche exhaustive. Ainsi, les cryptographes sont assurés qu'aucun autre algorithme plus efficace ne viendra mettre à mal la cryptographie symétrique actuelle. L'utilisation de schémas de cryptographie symmétrique disposant de tailles de clés d'au moins 200 bits (tels qu'AES256) permettra donc toujours de garantir au moins 100 bits de sécurité face à un attaquant disposant d'un ordinateur quantique et effectuant une recherche exhaustive.

Concernant la recherche de collisions dans les fonctions de hachage, une variante de l'algorithme de Grover nommée BHT [20] (du nom de ses trois inventeurs, Brassard, Høyer et Tapp) permet d'obtenir des collisions dans une fonction de hachage en $n/3$ calculs de hashs, ceux-ci devant être effectués sur la circuiterie quantique. Cela représente une accélération substantielle mais pas inquiétante pour autant comparée aux $n/2$ opérations nécessaires aujourd'hui du fait du paradoxe des anniversaires [25]. En conséquence, de même que pour les algorithmes de chiffrement, utiliser des algorithmes de hachage ayant une taille de sortie supérieure à 300 bits permet d'assurer une sécurité post-quantique d'au moins 100 bits [3].

L'algorithme de Shor L'algorithme de Shor est généralement présenté comme un algorithme permettant de factoriser un grand nombre en nombres premiers. Fondamentalement, il utilise un algorithme quantique permettant de trouver une période dans une fonction périodique. Pour rappel, une fonction f périodique de période T est telle que $f(x+T) = f(x)$ en tout point x de l'ensemble de départ.

L'algorithme de recherche de période nécéssite $log_2(L)$ qubits pour déceler une période de longueur L. Dans les grandes lignes, L valeurs

consécutives de l'espace de départ de f sont superposées dans les qubits, sur lesquelles sont appliquées f. Une transformée de Fourier (quantique) est appliquée sur l'état pour obtenir une représentation fréquentielle de la fonction initiale. Lors de la mesure finale de l'état, le résultat obtenu correspondra à la fréquence "la plus présente" dans la fonction, avec une bonne probabilité. La période pourra ensuite être déduite aisément.

Cet algorithme peut par exemple être utilisé pour factoriser un grand nombre n en nombres premiers en cherchant l'ordre d'un entier aléatoire dans le groupe multiplicatif des entiers modulo n, et donc la période de la fonction $f : x \mapsto a^x[n]$ pour un entier a tiré aléatoirement. En effet, si cette période est paire, alors $a^{r/2} + 1$ et $a^{r/2} - 1$ partagent un facteur avec l'entier n initial. Il suffit ensuite de calculer le PGCD de n et de chacune de ces deux quantités pour obtenir un ou plusieurs facteurs de n.

De même, cette routine peut être utilisée pour résoudre efficacement le problème du logarithme discret et ainsi mettre à mal la construction de Diffie-Hellman pour l'échange de clés. Il apparaît donc nécessaire de construire des algorithmes de cryptographie fondés sur d'autres problèmes mathématiques, ne pouvant pas être résolus par une recherche de période.

1.2 Les implications dès aujourd'hui : *Store now, decrypt later* et certaines signatures

Ces algorithmes s'exécutant sur des ordinateurs quantiques, et ceux-ci n'ayant pas encore atteint un niveau de perfectionnement suffisant pour dérouler les algorithmes mentionnés, un lecteur avisé est en droit de se demander pourquoi s'y intéresser dès maintenant. En plus de l'anticipation nécessaire pour construire les schémas de cryptographie de demain, une très bonne raison de migrer dès aujourd'hui réside dans le problème *Store now, decrypt later* : un attaquant capturant aujourd'hui une communication composée d'une phase d'échange de clés (pré-quantique) puis d'envois de données chiffrées à l'aide de la clé partagée, pourra attaquer hors-ligne l'échange de clés pré-quantique lorsque la puissance de calcul quantique sera suffisamment avancée. Une fois la clé partagée recouvrée, les messages échangés pourront être déchiffrés par l'attaquant.

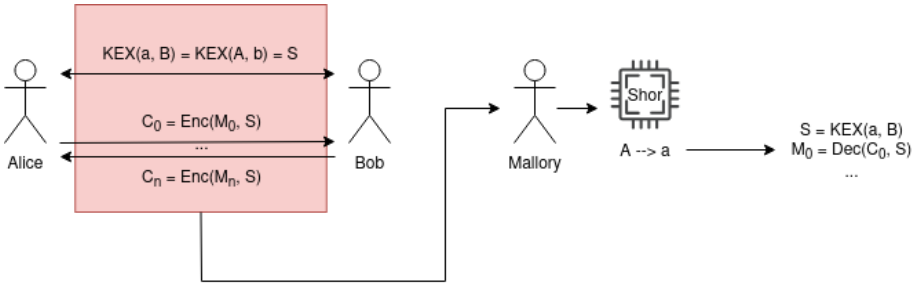

Fig. 1. Illustration du problème *Store now, decrypt later*

Cette menace démontre l'importance de sécuriser les mécanismes d'échange de clé dès aujourd'hui, en utilisant des algorithmes résistants aux attaques quantiques évoquées plus haut.

Concernant les algorithmes de signature (tels qu'ECDSA), l'impact dépend de la durée de vie et d'utilisation de celles-ci. Étant en général vérifiées peu de temps après avoir été émises[3] et n'étant plus réutilisées par la suite dans la plupart des cas, la menace quantique semble encore lointaine. Cependant, certaines signatures doivent conserver leur valeur de vérité pendant un temps plus long, par exemple des signatures de versions logicielles. Pour ces signatures, utiliser des mécanismes robustes face aux ordinateurs quantiques dès aujourd'hui est alors essentiel, afin d'éviter la falsification de celles-ci à l'avenir.

Les autres facettes de la cryptographie asymétriques telles que la cryptographie à seuil étant des sujets plutôt de niche et, pour certaines comme le schéma de partage de secret de Shamir, peu impactées par les algorithmes quantiques actuels, celles-ci ne seront pas traitées dans ce papier.

2 Comment passer à un monde "post-quantique" ?

2.1 Un problème de confiance

Comme vu dans la partie précédente, une transition importante est requise pour sécuriser les échanges de clés et les signatures vis-à-vis des algorithmes quantiques. Depuis des décennies, le monde académique propose des alternatives aux cryptosystèmes traditionnels, fondant ces nouveaux algorithmes sur des problèmes que l'on espère difficiles à résoudre à l'aide d'ordinateurs classiques comme quantiques. Certains de ces algorithmes

[3] par exemple, lors d'échanges TLS, les signatures sont vérifiées au cours des échanges

ont été standardisés très récemment (2024 et 2025), à la suite d'une compétition organisée par le NIST [4] [18] : ML-KEM [14] (dérivé de *Kyber*), ML-DSA [15] (dérivé de *CRYSTALS-Dilithium*), SLH-DSA [16] (dérivé de *SPHINCS+*), et HQC [17].

Bien que standardisés récemment à la suite d'études approfondies, ces "nouveaux" algorithmes ne sont pas l'objet d'une confiance absolue au sein de la communauté scientifique. En effet, les cryptanalystes ne disposent pas du même niveau de recul sur ces "nouveaux" problèmes, en comparaison d'algorithmes comme RSA ou ECDH étudiés depuis bientôt 50 ans et toujours considérés sûrs. L'été 2022 a par exemple vu apparaître une attaque s'exécutant sur un ordinateur classique permettant de casser la sécurité de SIDH, l'un des algorithmes proposés initialement pour des échanges de clés plus sûrs. La crainte de voir une telle situation se reproduire pour d'autres constructions est donc présente dans l'écosystème.

Ainsi, un algorithme idéal présenterait une sécurité *au moins aussi élevée* et éprouvée que les algorithmes "classiques" vis-à-vis d'un attaquant disposant d'un ordinateur classique, tout en disposant d'une sécurité *au moins aussi élevée* que les algorithmes "post-quantiques" vis-à-vis d'un attaquant disposant d'un ordinateur quantique. Un tel algorithme peut s'obtenir via l'hybridation, abordée dans la prochaine partie.

Famille	**Sécurité pré-Q**	**Sécurité post-Q**
Classiques (Courbes elliptiques, DH, ...)	+++	-
Nouveaux (Lattices, Codes, Hashs, ...)	++	++

Tableau 1. Confiance en la sécurité pré-quantique et post-quantique des différents algorithmes

L'ANSSI accompagne les acteurs français dans cette transition en publiant des recommandations claires et détaillées [2] [4].

2.2 L'hybridation

Signatures Dans le cas des signatures, la question de l'hybridation peut se résoudre rapidement ; il est aisé de se convaincre que la concaténation des signatures permet d'obtenir une sécurité maximale. Plus formellement, en considérant n algorithmes de signatures, l'algorithme suivant est au moins aussi sécurisé que le meilleur d'entre eux en toute situation :

[4] National Institute of Standards and Technologies, institut américain responsable des standards dans la tech

```
procedure CombiKeyGen
    for i = 1 to n do
        (S_i, P_i) ← KeyGen_i()
    S ← (S_1, S_2, ..., S_n)
    P ← (P_1, P_2, ..., P_n)
    return (S, P)
procedure CombiSign(m, S = (S_1, S_2, ..., S_n))
    for i = 1 to n do
        sig_i ← Sign_i(m, S_i)
    sig ← (sig_1, sig_2, ..., sig_n)
    return sig
procedure CombiVerify(m, sig = (sig_1, ..., sig_n), P = (P_1, ..., P_n))
    for i = 1 to n do
        if Verify_i(m, sig_i, P_i) = false then
            return false
    return true
```

Algorithme 1. Combineur pour signatures

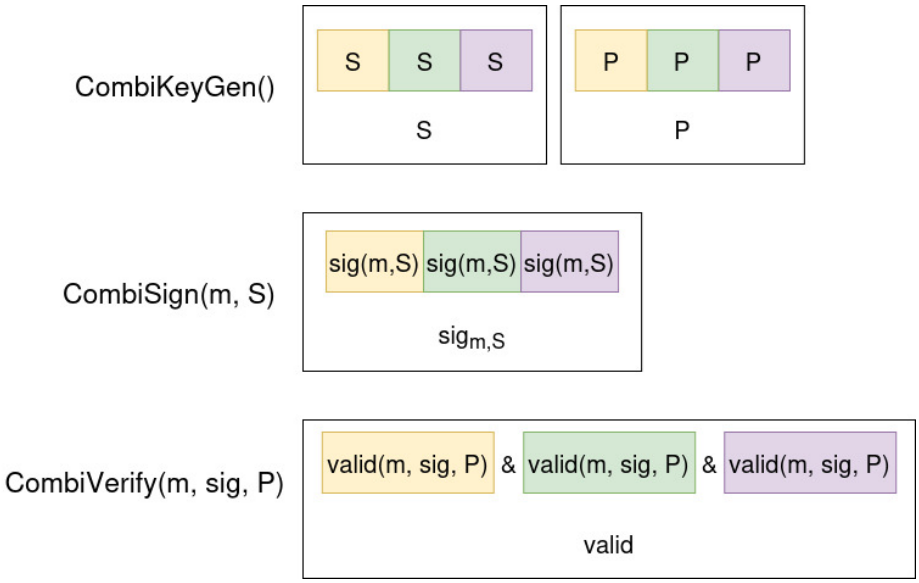

Fig. 2. Combineur de signature, concaténant simplement les valeurs des différents algorithmes

Pour forger un couple (m, sig) valide vis-à-vis de $CombiVerify$, un attaquant devra forger une signature valide pour chaque algorithme. En effet, si une seule vérification échoue, alors la signature combinée sera invalide. En combinant un algorithme "classique" et un ou plusieurs al-

gorithmes "post-quantiques" par cette méthode, il est donc possible de construire un schéma de signature digne de confiance face aux attaques actuelles et futures.

Le coût en performance d'une telle méthode est raisonnable ; les signatures peuvent être calculées en parallèle. De même, il semble difficile de compresser davantage la signature résultante. En effet, les algorithmes de signatures nécessitent généralement de réaliser des opérations sur la signature brute lors de la vérification de celle-ci. Pour vérifier chacune des sous-signatures du combineur, il faut donc que le vérifieur dispose des données brutes de chacune d'entre elles. Ce combineur simple est donc au plus proche de l'optimum générique, et semble accepté par la communauté comme en témoigne l'ANSSI [2].

Échanges de clés Concernant les échanges de clés, le problème est plus complexe. Commençons par définir une interface générique, communément nommée *Key Encapsulation Mechanism* ("KEM"), nous permettant d'abstraire un algorithme d'échange de clés. Cette interface comprend trois fonctions :

— $KeyGen() \rightarrow (S, P)$, générant une bi-clé à partir d'un générateur d'aléa
— $Encaps(P) \rightarrow (ss, caps_P)$, générant un secret partagé ss simplement à l'aide de la clé publique de l'interlocuteur. Un générateur aléatoire est utilisé en interne pour assurer la sécurité du secret. Le secret est encapsulé dans la "capsule" $caps_P$, ouvrable uniquement via la partie privée associée à P. La capsule peut donc être transmise à l'interlocuteur sur un canal non-sécurisé
— $Decaps(caps_P, S) \rightarrow ss$, "décapsulant" le secret partagé à l'aide de la partie privée de la clé associée à P.

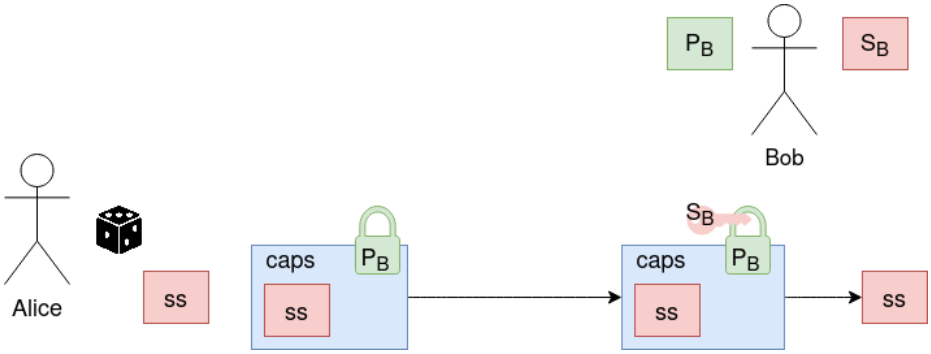

Fig. 3. Key Encapsulation Mechanism

Cette construction ressemble au premier abord à un simple algorithme de chiffrement asymétrique, dans lequel Alice décide d'une clé et la partage de manière chiffrée à Bob. En pratique, la clé publique peut intervenir dans la définition du secret partagé. Par exemple, un échange de clés via le protocole de Diffie-Hellman peut être représenté par un KEM de la manière suivante :

— $KeyGen()$: tirer a aléatoire, retourner $(K_{priv} = a,\ K_{pub} = A = g^a[p])$
— $Encaps(K_{pub})$: tirer b aléatoire, retourner $(ss = (K_{pub})^b[p],\ caps = B = g^b[p])$
— $Decaps(caps, K_{priv})$: retourner $caps^{K_{priv}}[p]$

Un article de 2018 par Giacon, Heuer et Poettering [11] introduit une notion clé pour quantifier le niveau de sécurité d'un combineur de KEMs : l'indistinguabilité du secret partagé. Un attaquant compromettant une partie des KEMs (capable de déchiffrer des capsules ne lui étant pas destinées) ne doit pas être capable de distinguer le secret partagé d'une valeur aléatoire. Ici, si les secrets de chaque KEM sont simplement concaténés et qu'un attaquant parvient à déchiffrer l'un d'entre eux, il connaîtra avec certitude une partie des bits de la clé résultante. Une construction simpliste n'est donc pas satisfaisante.

De manière plus subtile, une combinaison des secrets de type $ss = H(ss_1||ss_2||...||ss_n)$ n'est pas non plus idéale. En effet, pour certaines combinaisons de KEM et fonction de hachage, un attaquant ayant accès à un oracle de déchiffrement réalisant les décapsulations et la combinaison des secrets pourrait être capable de distinguer la clé résultante vis-à-vis d'une valeur aléatoire [11]. Plus de détails sur une telle attaque sont présentés dans le papier mentionné.

La communauté s'accorde cependant pour dire qu'en concaténant les capsules à la combinaison précédente (donc $ss = H(ss_1||ss_2||...||ss_n||caps_1||caps_2||...||caps_n)$), la construction résultante est robuste. Celle-ci est appelée CatKDF [9] [4], et représente un bon candidat générique pour combiner plusieurs KEMs. D'autres constructions génériques robustes ont aussi été proposées, ainsi que certaines constructions optimisées pour des KEMs spécifiques telles X-Wing [6] pour combiner efficacement X25519 et ML-KEM (Kyber). Celles-ci ne seront toutefois pas abordées dans ce papier.

3 Notre montée en compétence sur le sujet

Au sein de Synacktiv, un sondage informel a permis de déceler une dizaine de consultants intéressés par le sujet et motivés pour monter en compétence dans le domaine. Ceux-ci possédaient un bagage technique et des compétences variées : certains ont des appétences pour la théorie et les mathématiques, d'autres préfèrent étudier les implémentations logicielles d'algorithmes existants, et d'autres encore sont davantage intéressés par l'analyse matérielle. Cependant, aucun des profils désignés n'est réellement "chercheur" au sens académique du terme, Synacktiv restant une entreprise d'audit et de conseil.

Quelques recherches avaient préalablement été menées en interne sur le sujet de la cryptographie post-quantique, pour mettre le pied à l'étrier. De ces premiers efforts avaient découlé un article, paru en 2021 sur le blog de Synacktiv [10] et clarifiant les principes associés à l'informatique quantique, ainsi que l'état de la menace et de la défense face à de tels ordinateurs, sans entrer dans le détail de la théorie associée.

Il a d'abord été décidé de poursuivre ces travaux, en lisant des publications académiques pour poursuivre cette montée en compétence. Quelques consultants courageux se sont donc attelés à la lecture de publications sur différents algorithmes de cryptographie post-quantique. Cependant, l'équipe n'étant pas habituée à se former à l'aide de publications académiques, il a fallu se rendre à l'évidence : l'effort aurait été trop important, et l'expérience assez difficile et démotivante pour les consultants impliqués.

C'est alors qu'il a été décidé de participer à la PQC Spring School 2024,[5] une semaine de cours dédiés à une introduction au domaine de la cryptographie post-quantique à destination de doctorants en cryptographie. Cet évènement était organisé par Quantum-Safe Internet,[6] un projet

[5] PQC Spring School 2024, `https://pqc-spring-school.nl/`
[6] Quantum-Safe Internet, `https://quantum-safeinternet.com/`

financé par l'Union Européenne. Après une introduction sur la cryptographie classique, différents cours ont ainsi été donnés sur des notions utiles en cryptographie post-quantique : les oracles quantiques, les codes correcteurs d'erreurs, la cryptographie basée sur les algorithmes de calcul d'empreinte (hashs)... Cette approche a été particulièrement efficace pour dynamiser la formation de l'équipe, car les chercheurs présents transmettaient une intuition visuelle sur le fonctionnement des algorithmes, ce qui rendait plus facile à appréhender la théorie associée. De plus, le fait d'assister à ces présentations en groupe permettait d'en discuter au sein du groupe durant les pauses, pour s'assurer de la compréhension de la plus grande partie du contenu énoncé.

A l'issue de cette semaine, l'équipe avait davantage d'outils pour comprendre les concepts et les différents problèmes autour de la cryptographie post-quantique. Il s'agissait d'un travail nécessaire, permettant d'aborder plus sereinement la lecture de publications académiques pour, d'une part, peaufiner l'état de l'art en interne et d'autre part, être prêts à "décrypter" de nouveaux concepts et de nouveaux articles qui seraient publiés à l'avenir.

La rédaction de plusieurs articles de blog a été entamée à la suite de cette formation, dans le but de restituer de manière structurée les connaissances acquises pendant les étapes précédentes et les partager à un plus grand nombre. Le public cible est alors un lecteur qui possède des notions de mathématiques et de cryptographie classique, mais qui découvre les notions de cryptographie post-quantique. Ce travail de vulgarisation a aussi permis de déceler d'éventuels points d'incertitudes restants dans la compréhension des algorithmes et des enjeux, afin de poursuivre l'autoformation. Au total, ce travail de formation représente environ 200 jours sur une période de 4 ans, répartis sur la dizaine de personnes intéressées.

Cette présentation s'inscrit aussi dans le cadre de la transmission ; le domaine de la cryptographie "post-quantique" est victime d'un fort préjugé associant "post-quantique" à "très compliqué". Synacktiv cherche donc à faire tomber cette idée reçue, dans l'objectif de motiver d'autres professionnels de la cybersécurité à se former dans ce sous-domaine de la cryptographie, qui en deviendra une part centrale dans les années à venir.

4 Des exemples d'algorithmes

4.1 SLH-DSA, un algorithme de signature post-quantique

Les fonctions de hachage cryptographique sont considérées, au même titre que la cryptographie symétrique, comme globalement robustes face

aux ordinateurs quantiques (voir le paragraphe 1.1 sur l'algorithme de Grover). De plus, celles-ci sont déjà démocratisées, de nombreuses bibliothèques cryptographiques proposant une implémentation des fonctions de hachage usuelles. Ainsi, en tant que famille de fonctions difficiles à inverser, les fonctions de hachage représentent un bon candidat pour construire un algorithme de signature asymétrique.

Dans la suite de ce paragraphe, H désignera une fonction de hachage robuste (SHA-3 par exemple).

Signer un seul message d'un seul bit avec une fonction de hachage Commençons par un cas simple, en supposant qu'Alice souhaite signer un message d'un seul bit ($m = 0$ ou $m = 1$). Pour cela, elle peut commencer par choisir deux chaînes aléatoires, disons "PremierSecret" et "SecondSecret". Elle publie alors le couple $(H_0 = H(\text{"}PremierSecret\text{"}), H_1 = H(\text{"}SecondSecret\text{"}))$, qui forme sa clé publique. Pour signer le message $m = 0$, elle révèle simplement "PremierSecret". Tout le monde peut ensuite vérifier que $H(\text{"}PremierSecret\text{"}) = H_0$, confirmant qu'Alice connaissait une pré-image de H_0. Pour rappel, le calcul de pré-image est considéré comme difficile et hors de portée d'un attaquant.

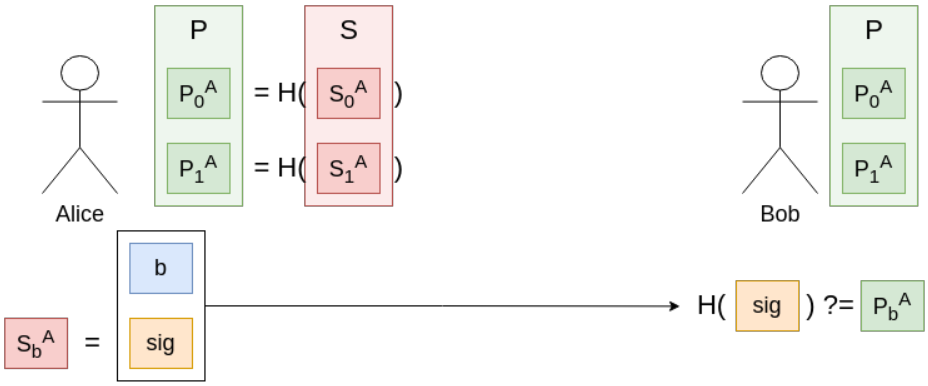

Fig. 4. Signature d'un message d'un bit

Cette construction, bien que très limitée, montre l'idée intuitive derrière ces algorithmes de signature : une clé publique composée d'images de secrets par H, et des signatures correspondant à leurs pré-images.

Les chaînes de hashs Pour signer des messages plus longs ($0 \leq |m| < n$, avec $n \geq 2$ et $|m|$ représentant la longueur en bits du message m), il est

possible d'utiliser une approche par "chaîne". Plutôt que de définir un secret par valeur possible du message (comme dans l'exemple précédent où notre bit avait deux valeurs possibles, et où deux secrets étaient générés), un seul secret est ici généré. Pour signer le message m, on applique simplement m fois de suite l'opération de hachage sur le secret.

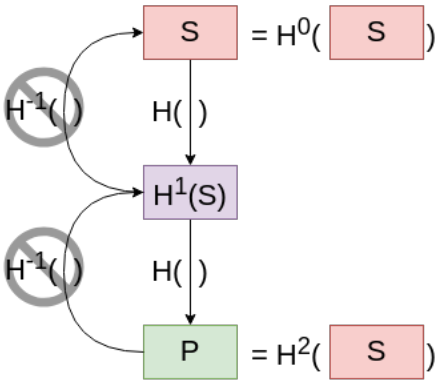

Fig. 5. Calcul de la clé publique à partir de la clé privée via la chaîne de hashs (longueur 2 ici)

Ainsi, en choisissant une unique valeur privée S, puis en la hachant 2^n fois pour obtenir la clé publique P, on obtient un potentiel nouveau schéma de signature. Une signature d'un message m serait alors $sig = H^m(S)$, vérifiable par $P \stackrel{?}{=} H^{2^n-m}(sig)$. En effet, si sig est correcte, alors $H^{2^n-m}(sig) = H^{2^n-m}(H^m(S)) = H^{2^n}(S) = P$. En somme, cela revient à avancer de m étapes sur la chaîne pour signer m, puis d'effectuer les $2^n - m$ étapes restantes pour arriver à la clé publique lors de la vérification.

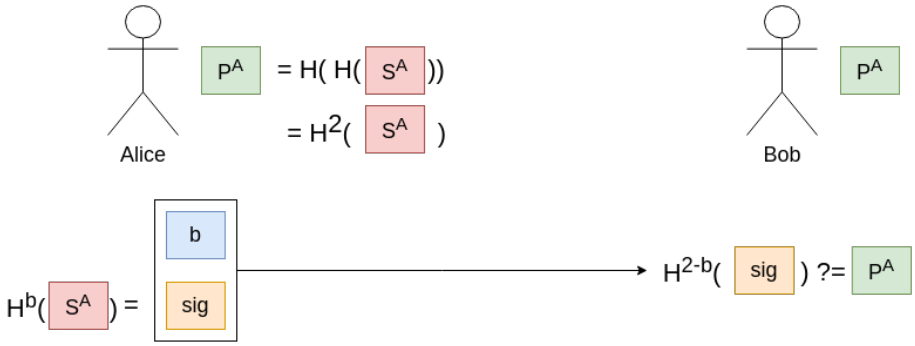

Fig. 6. Signature du message b ($b = 0$ ou $b = 1$) par une chaîne de hashs de longueur 2

Cependant, cela pose plusieurs problèmes :

— la complexité temporelle pour calculer et vérifier une signature est exponentielle en la taille du message ;
— un attaquant en possession de la signature d'un message m_1 serait en capacité de signer tout message $m_2 > m_1$ simplement en hachant $m_2 - m_1$ fois la signature de m_1. L'authentification et la non-répudiation des messages signés ne sont donc pas assurés par cette méthode.

Pour pallier le premier problème, il est possible de fixer une taille de chaîne raisonnable (par exemple $n = 4$ bits, pour obtenir des chaînes de longueur 16), de découper le message à signer en blocs de n bits, et de signer chaque bloc avec une clé différente (sur des chaînes parallèles, en quelque sorte). Ainsi, au lieu d'augmenter exponentiellement en temps, la complexité de signature augmente linéairement en temps et en espace avec la taille du message à signer.

Pour résoudre le second problème, une somme de contrôle doit être introduite dans le message à signer, de sorte qu'un pas en avant dans la chaîne d'un bloc de message corresponde nécessairement à un pas en arrière dans la chaîne d'un bloc de somme de contrôle. Une manière d'envisager cela est de signer en parallèle la valeur $2^n - 1 - m$. Cela correspond à avancer d'un certain nombre de pas sur une chaîne, et du complémentaire dans l'autre. Faire un pas de plus sur l'une des deux chaînes revient nécessairement à en faire un de moins sur l'autre, et nécessite donc la connaissance d'une pré-image.

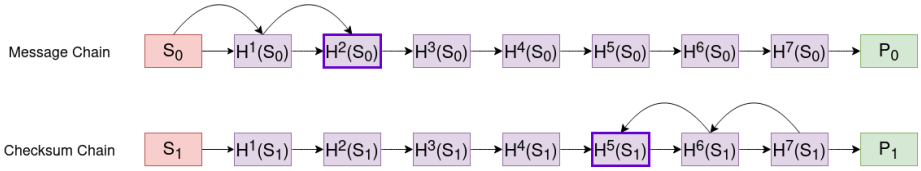

Fig. 7. Signature du message $m = 2$ sur deux chaînes

Des optimisations existent pour réduire la taille de la checksum (voir l'article sur le blog Synacktiv [12]), mais ne seront pas couvertes dans cette présentation pour des raisons de temps.

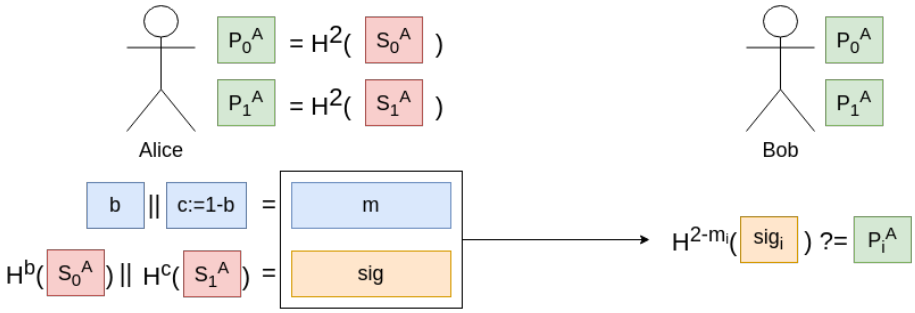

Fig. 8. Signature d'un message et de sa checksum via des chaînes de hachage

Attention toutefois, si Alice souhaitait signer un second message avec ces mêmes clés, la publication de cette seconde signature révélerait un maillon supplémentaire dans chacune des chaînes, et ainsi un attaquant pourrait générer des signatures "intermédiaires" (ie. dont les blocs se situent après des blocs de signatures révélées, tant pour le message que pour la checksum). Il faut donc qu'Alice regénère une paire de clés pour chaque nouvelle signature, en publiant à chaque fois la clé publique associée.

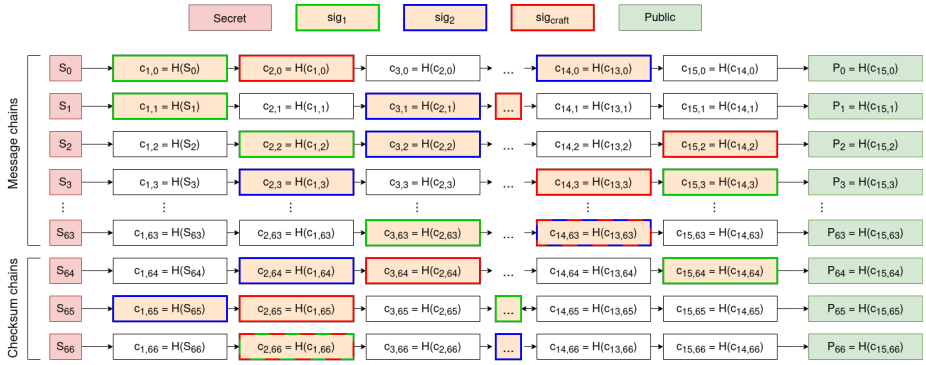

Fig. 9. Génération d'une signature frauduleuse sig_{craft} à partir de deux signatures légitimes.

La construction décrite dans ce paragraphe correspond dans les grandes lignes à l'algorithme de signature à usage unique de Winternitz, abrégé WOTS (Winternitz One-Time Signature) dans la suite de cet article.

Réduire les tailles de clés Plutôt que de publier chaque clé une par une, une solution consiste à créer un arbre binaire dont les feuilles sont des clés publiques.

Cette structure, appelée arbre de Merkle [22], permet d'alléger fortement la clé publique (seule la racine de l'arbre est publiée), en alourdissant peu les signatures.

Pour authentifier une feuille de cet arbre, il suffit de dévoiler les valeurs intermédiaires au calcul de la racine. Les valeurs révélées seront alors concaténées à droite ou à gauche selon si le l-ième bit de i est à 0 ou 1, avec l la hauteur dans l'arbre et i l'indice de la feuille à authentifier. Par exemple, pour authentifier $u = P_1$, il faut révéler P_0, P_{23} et P_{4567} afin que toute personne souhaitant vérifier que u est bien la feuille d'indice $i = 1$ dans l'arbre puisse suivre le protocole suivant :

— Recalculer $P'_{01} = H(P_0||u)$ (l'ordre de la concaténation se déduit du bit 0 de i, qui vaut 1)
— Recalculer $P'_{0123} = H(P'_{01}||P_{23})$ (l'ordre de la concaténation se déduit du bit 1 de i, qui vaut 0)
— Recalculer $P'_{01234567} = H(P'_{0123}||P_{4567})$ (l'ordre de la concaténation se déduit du bit 2 de i, qui vaut 0)
— Vérifier que $P'_{01234567} \stackrel{?}{=} P$, avec P la racine (publique) de l'arbre.

Notons que les valeurs intermédiaires révélées ne compromettent pas de clés privées.

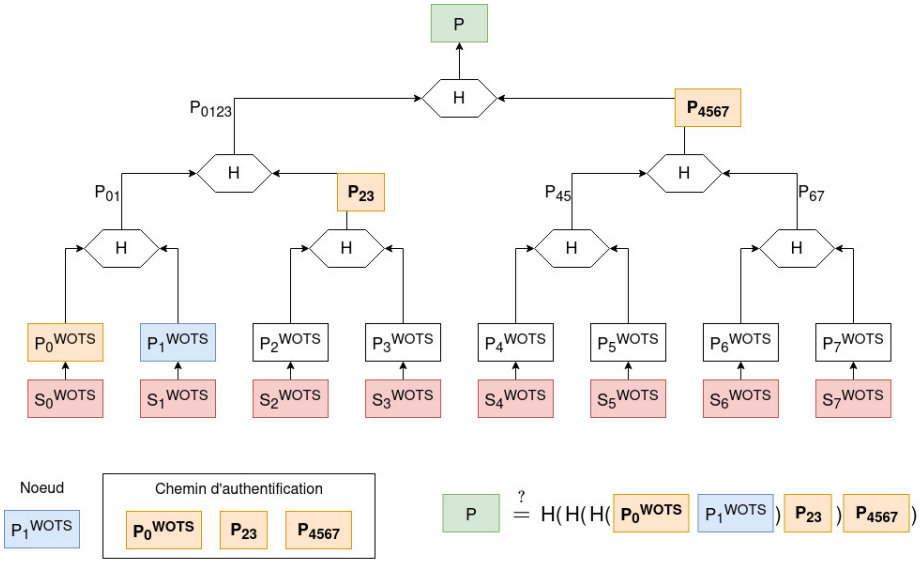

Fig. 10. Authentification d'une feuille dans un arbre de Merkle

Ainsi, en publiant uniquement la racine de l'arbre, il est possible d'authentifier chacune des clés générées par Alice. Il suffira d'adjoindre le chemin d'authentification de la clé publique associée à chaque signature pour démontrer que la clé privée utilisée appartient bien à Alice.

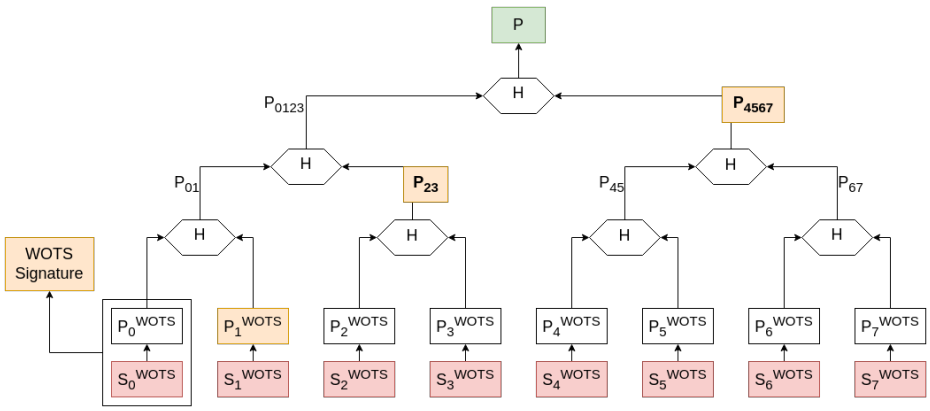

Fig. 11. Signature WOTS dans un arbre de Merkle de hauteur 3

Parallèlement, il est possible d'alléger la clé privée en se servant d'une fonction de hachage comme d'un générateur aléatoire. En effet, via une fonction de dérivation de clés (KDF), par exemple PBKDF ou HMAC, n secrets peuvent être générés à partir d'un unique secret initial et d'un compteur.

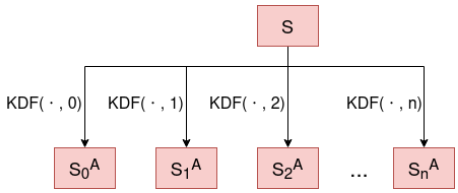

Fig. 12. Génération des secrets à partir d'un unique secret et d'une fonction de dérivation de clés H, avec un compteur

XMSS Le schéma de signature présenté ci-dessus représente essentiellement le schéma de signature de Merkle (MSS) [24]. Quelques optimisations permettant d'assurer une meilleure robustesse en cas d'attaque sur les fonctions de hachage amènent à XMSS [13] (eXtended MSS, standardisé par le NIST via la publication SP 800-208 [19]), mais celles-ci ne seront pas détaillées dans la présentation. Pour davantage de détails, se référer à l'article associé sur le blog de Synacktiv [12].

L'essentiel de la sécurité de cet algorithme repose sur le fait que les feuilles de l'arbre de Merkle ne doivent pas être utilisées plus d'une fois. Cela implique le maintient d'un "état" de chaque feuille (utilisée ou non), ou plus simplement de l'identifiant de la dernière feuille utilisée, en parallèle de la clé privée. Ainsi, XMSS doit uniquement être utilisé dans des applications **séquentielles**, **mono-threadées**, en avançant le compteur d'utilisation **avant** la publication d'une signature (pour s'assurer qu'aucune signature non-comptabilisée n'est publiée) et avec une attention particulière à l'état des clés lors d'une reprise sur sauvegarde.

Des signatures à usage "quelques fois" ("few-times") Dans le but de contourner les restrictions associées aux schémas de signature à usage unique, de nouveaux schémas ont été proposés, permettant des clés réutilisables *un petit nombre de fois*. Pour cela, l'algorithme HORS [5] (l'un des schémas de signature "few-times") fonctionne comme suit :

procedure INITIALISATION DES PARAMÈTRES DE L'ALGORITHME
  $t \leftarrow 1024$ ▷ puissance de 2
  $k \leftarrow 16$ ▷ entier quelconque, pas trop grand
  Clé privée : $S \leftarrow [S_0, S_1, ..., S_{t-1}]$ ▷ liste de valeurs aléatoires de taille t
  Clé publique : $P \leftarrow [F(S_0), F(S_1), ..., F(S_{t-1})]$ ▷ image des valeurs par une fonction à sens unique F
procedure SIGNER(m)
  $h \leftarrow H(m)$ ▷ Hacher le message m avec une fonction de hachage H
  $h_i \leftarrow$ découper(h, k) ▷ Découper h en k blocs de taille $\log_2(t)$ bits
  $sig \leftarrow [S[h_1], S[h_2], ..., S[h_k]]$ ▷ Construire la signature en sélectionnant les S_{h_i}
  return sig
procedure VÉRIFIER(m, sig, P)
  $h \leftarrow H(m)$ ▷ Hacher le message m
  $h_i \leftarrow$ découper(h, k) ▷ Découper h en k blocs
  for $i = 1$ **to** k **do**
    if $F(sig_i) \neq P[h_i]$ **then**
      return `faux` ▷ La signature est invalide
  return `vrai` ▷ La signature est valide

Algorithme 2. Signature et Vérification HORS

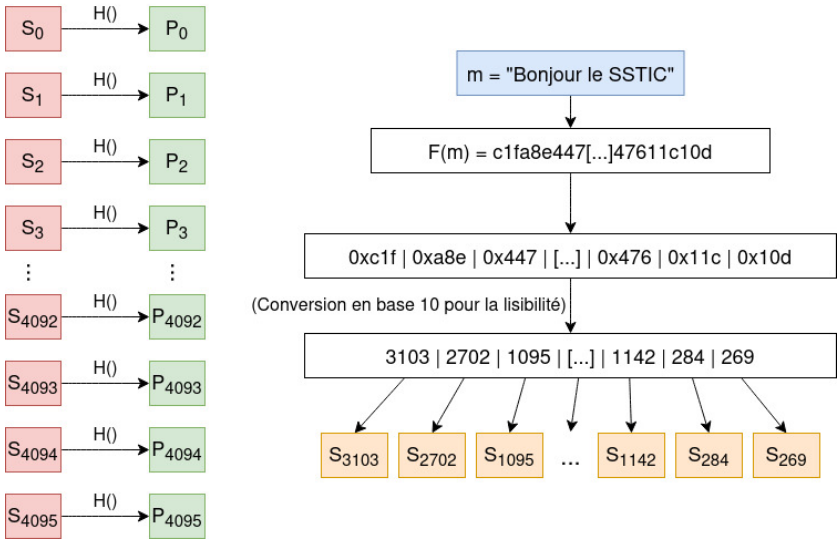

Fig. 13. Schéma de signature HORS

Dans ce schéma de signature et avec les paramètres proposés, falsifier une signature avec la connaissance d'une unique signature préalable demanderait à un attaquant de trouver un message composé des mêmes blocs h_i que le message initialement signé et dont les portions de clé privée

ont été révélées. De même, en connaissant n signatures, au maximum $k \times n$ blocs de clé privée ont été révélés. Forger une signature est donc équivalent à trouver un message m dont le condensat (hash) est composé des indices de ces $k \times n$ blocs. En supposant la fonction de hachage assimilable à un oracle aléatoire, trouver un tel message se fait avec une probabilité $p_{n,k,t} = (\frac{k \times n}{t})^k$. Le nombre de bits de sécurité peut donc être calculé, et est représenté par la courbe ci-dessous en fonction du nombre de signatures publiées pour $t = 2^{10}$ et $k = 16$.

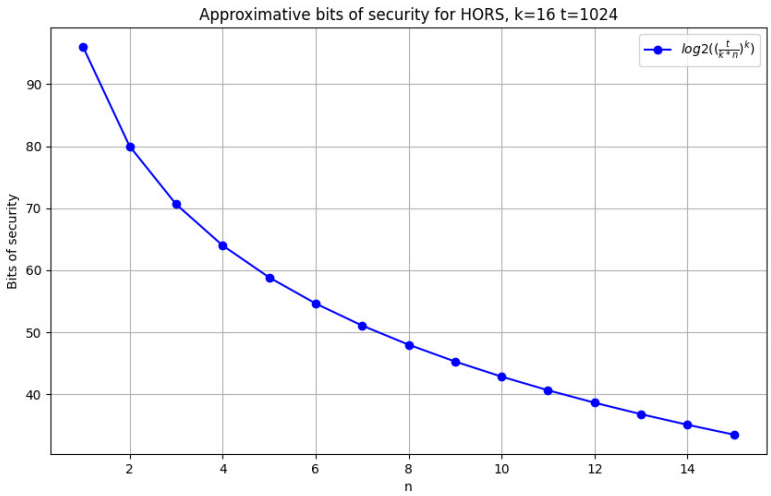

Fig. 14. Affaiblissement de la sécurité de HORS après n signatures

On voit donc qu'avec des paramètres plus grand, il pourrait être possible d'obtenir une sécurité correcte malgré la publication de quelques signatures générées par la même clé.

SPHINCS+ / SLH-DSA SLH-DSA, algorithme standardisé par le NIST (via la publication FIPS 205 [16]) utilise FORS, un schéma de signature "few-times" dérivé de HORS, pour réaliser les signatures. Afin de grouper un maximum de paires de clés FORS dans une même clé SLH-DSA, une structure dite "multi-tree" est construite afin de combiner toutes ces clés au moyen de signatures XMSS. Une telle structure peut être vue comme une infrastructure à clé publique, avec une autorité de certification racine (l'arbre XMSS du niveau le plus haut du multi-tree),

dont la clé publique (la racine de l'arbre) est distribuée. Cet arbre permet de signer les clés publique de sous-arbres (représentant des autorités de certification intermédiaires), permettant eux-mêmes de signer d'autres clés publiques d'arbres intermédiaires, pour enfin arriver aux arbres signant les clés FORS. Les signatures SLH-DSA sont donc composées d'une signature FORS et du chemin d'authentification (avec les signatures intermédiaires) permettant de remonter à la racine du multi-tree.

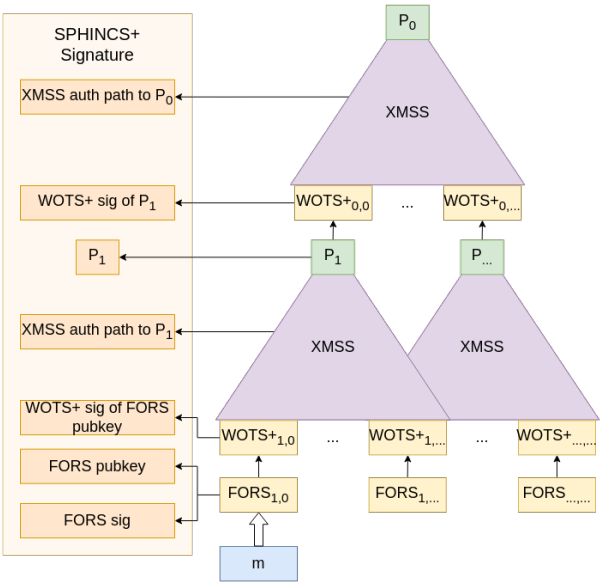

Fig. 15. Signature avec SLH-DSA

Ainsi, avec autant de paires de clés FORS et une contrainte plus souple sur la réutilisation de clés, SLH-DSA se défait de la contrainte qu'est la gestion d'un état. À la place, la paire de clé à utiliser est tirée aléatoirement pour chaque signature. Ainsi, il est possible de calculer la probabilité qu'une même paire de clés FORS soit réutilisée "un trop grand nombre de fois" après n signatures SLH-DSA, et d'en déduire la durée de vie d'une clé SLH-DSA en fonction du nombre de clés FORS et de leur durée de vie individuelle.

Ici, plusieurs points d'attention sont à observer :

— le générateur aléatoire ne doit pas être biaisé, afin que toutes les clés s'usent équitablement.

— les signatures des arbres intermédiaires sont réalisées à partir d'algorithmes de signature à usage unique (signature de la racine d'un arbre XMSS par une feuille (clé WOTS+) d'un arbre XMSS du niveau supérieur). Ainsi, il est important de s'assurer qu'une même clé intermédiaire ne puisse signer qu'une seule valeur, par exemple en stockant la valeur de la racine de l'arbre de niveau inférieur. En particulier, en cas d'injection de faute, il pourrait être possible de signer une valeur différente avec une même clé intermédiaire.

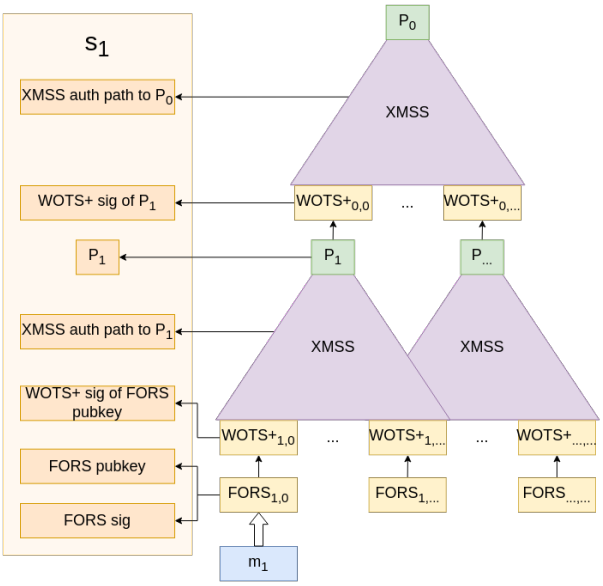

Fig. 16. Première signature (légitime) avec SLH-DSA

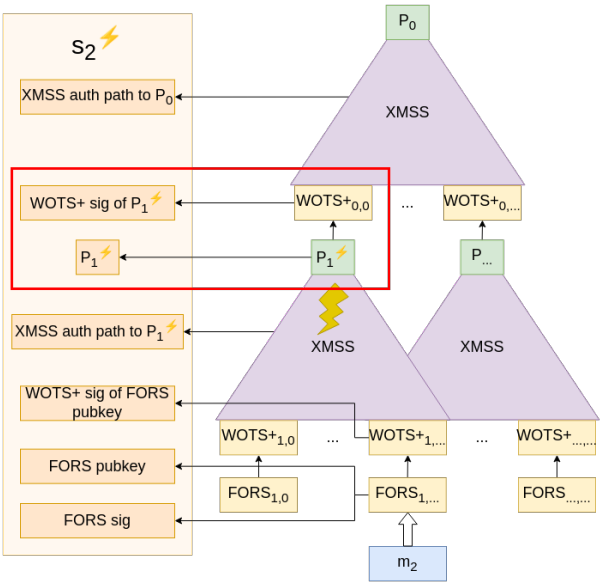

Fig. 17. Seconde signature (fautée) avec SLH-DSA

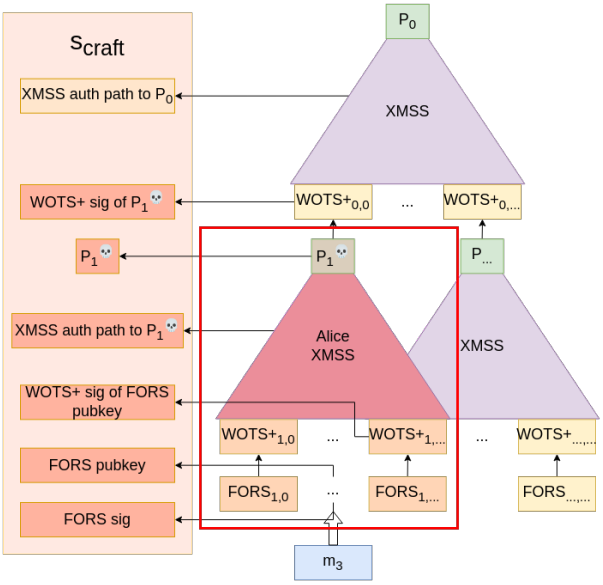

Fig. 18. Signature forgée résultante

Dans les schémas ci-dessus, une première signature légitime S_1 est générée. Lors d'une seconde signature provenant du même sous-arbre que S_1, une faute est injectée lors du calcul de la racine de ce sous-arbre. Ainsi, la feuille $WOTS^+$ de l'arbre de niveau supérieur qui avait précédemment signé P_1 signe désormais P_1^{volt}. Cela contredit le principe de "signature à usage unique", et permet de forger des signatures grâce à l'attaque présentée dans la figure 9. Il devient donc faisable pour un attaquant de forger un nouveau sous-arbre, de racine P_1^{skull} signable, puis de signer des messages arbitraires via les clés FORS aux feuilles de ce nouvel arbre.

4.2 Kyber, un algorithme d'échange de clés post-quantique

Les schémas d'échange de clés à base de *lattices* (réseaux euclidiens) sont aujourd'hui considérés comme l'une des approches les plus prometteuses en cryptographie post-quantique. Ils reposent sur des problèmes mathématiques (tels que *Learning With Errors* - LWE [21], ou ses variantes sur les anneaux - RLWE [26], MLWE [1], . . .) réputés difficiles à résoudre, même pour un ordinateur quantique, bien qu'aucune preuve formelle sur l'inexistance d'un algorithme quantique "efficace" n'ait été présentée. À l'instar des signatures basées sur les fonctions de hachage, l'idée générale est de s'appuyer sur des opérations qui sont simples à réaliser, tout en rendant leur inversion (par un attaquant) extrêmement coûteuse.

Dans cette partie, nous allons présenter les grandes lignes du mécanisme d'échange de clés par *lattices* (ou “réseaux"), en se focalisant sur l'intuition mathématique et les points d'attention, plutôt que sur les démonstrations formelles.

Le problème "Learning With Errors" (LWE) Le problème LWE, et ses variantes, constituent le socle de nombreux schémas cryptographiques post-quantiques.

— **Principe général :** On considère un espace vectoriel (par exemple $\mathbb{Z}_q^n$) et un *secret* s, vecteur dans cet espace. Le problème LWE consiste, pour un attaquant, à retrouver s à partir de plusieurs équations linéaires entachées d'un bruit aléatoire (les erreurs). Concrètement, on publie des paires (a, b) où $b = \langle a, s\rangle + e \mod q$, avec e un bruit discret.

— **Pourquoi est-ce difficile ?** Sans bruit ($e = 0$), il suffirait de résoudre un système linéaire pour retrouver s. Avec le bruit, le problème devient bien plus ardu.

— **Post-quantique ?** Les meilleures techniques connues (algorithmes classiques ou quantiques) pour retrouver s sont d'une complexité

exponentielle en la dimension n et/ou en la taille du module q. Cela rend les schémas basés sur LWE ou RLWE particulièrement intéressants.

Schéma d'échange de clés Sur la base de LWE, il est possible de construire un schéma d'échange de clés qui fonctionne de manière similaire à un Diffie-Hellman classique, mais avec des opérations sur des *polynômes* ou des *vecteurs* et l'ajout d'un bruit aléatoire. Nous exposons ci-dessous l'idée générale, tout en simplifiant les notations :

1. **Paramètres publics :** choisir un entier n, un module (modulo pour les opérations) q, une distribution de bruit χ, et une matrice aléatoire A (commune aux deux parties).
2. **Génération de clé (Alice) :**
   — Alice génère un vecteur secret s_A selon χ.
   — Elle calcule la partie publique $p_A = A \cdot s_A + e_A \mod q$, où e_A est un bruit tiré selon χ.
   — La clé publique d'Alice est p_A, et sa clé privée est s_A.
3. **Génération de clé (Bob) :**
   — Bob génère son vecteur secret s_B et calcule sa partie publique $p_B = A \cdot s_B + e_B \mod q$.
   — La clé publique de Bob est p_B, et sa clé privée est s_B.
4. **Échange :**
   — Bob utilise la clé publique d'Alice pour calculer $k_B = \langle p_A, s_B \rangle \mod q$. Étant donné que $p_A = A \cdot s_A + e_A$, on obtient $k_B \approx \langle A \cdot s_A, s_B \rangle + \langle e_A, s_B \rangle \mod q$.
   — Alice utilise la clé publique de Bob pour calculer $k_A = \langle p_B, s_A \rangle \mod q$. Par analogie, $p_B = A \cdot s_B + e_B$, donc $k_A \approx \langle A \cdot s_B, s_A \rangle + \langle e_B, s_A \rangle \mod q$.
5. **Clé commune :**
   — En raison des produits scalaires, k_A et k_B sont proches l'un de l'autre. Des techniques de *reconciliation* (souvent des seuils et des arrondis) permettent de corriger l'impact du bruit et d'obtenir une même clé secrète commune k.

Exemple : Baby Kyber Pour illustrer le fonctionnement de Kyber de manière simplifiée, on considère un exemple de très petite taille (appelé ici « Baby Kyber »). Même si les valeurs numériques sont volontairement réduites pour des raisons pédagogiques (et ne sont donc absolument pas sécurisées), le principe reflète bien la logique du système Kyber réel.

Paramètres réduits

- **Le module** q est choisi très petit. Par exemple $q = 17$. Dans la version standard de Kyber, $q \approx 3000$.
- **Modulus polynômial** : on travaille avec des polynômes modulo $x^4 + 1$. Cela signifie qu'on ramène constamment le degré des polynômes à un maximum de 3 (les puissances ≥ 4 sont réduites modulo $x^4 + 1$).
- **Vecteurs de polynômes** :
  - Dans Baby Kyber, on manipule seulement quelques polynômes (de petites dimensions).
  - Dans le Kyber réel, la dimension (et la structure de la matrice $\mathbf{A}$) est bien plus grande, et des techniques de *compression* sont aussi employées.

Génération de clés

- **Clé secrète s** : on génère un petit vecteur de polynômes. Dans Baby Kyber, la clé secrète contient par exemple deux polynômes de coefficients « petits » :

$$\mathbf{s} = (-x^3 - x^2 + x,\ -x^3 - x).$$

- **Matrice publique A** : on choisit une matrice de polynômes aléatoires. Par exemple,

$$\mathbf{A} = \begin{pmatrix} 6x^3 + 16x^2 + 16x + 1 & 9x^3 + 4x^2 + 6x + 1 \\ 5x^3 + 3x^2 + 10x + 1 & 6x^3 + x^2 + 9x + 15 \end{pmatrix} \mod (x^4{+}1,\ 17).$$

- **Vecteur d'erreur e** : on génère également un vecteur de polynômes « petits ». Par exemple,

$$\mathbf{e} = (x^2,\ x^2 - x).$$

- **Clé publique t** : on calcule

$$\mathbf{t} = \mathbf{A}\,\mathbf{s} + \mathbf{e} \qquad \mod (x^4 + 1,\ 17).$$

- **Résultat** : la clé privée est donc $\mathbf{s}$. La clé publique comprend $\mathbf{A}$ et $\mathbf{t}$.

L'idée cruciale est qu'il est difficile (censément impossible en pratique pour de grands paramètres) de retrouver $\mathbf{s}$ en résolvant $\mathbf{t} = \mathbf{A}\,\mathbf{s} + \mathbf{e}$, compte tenu du bruit $\mathbf{e}$. Cela correspond au problème MLWE (Module Learning With Errors).

Chiffrement Pour chiffrer un message $\mathbf{m}$ (souvent un secret servant à générer une clé symétrique), on fait ce qui suit :

— On encode le message dans un polynôme $\mathbf{m}_b$ contenant les bits du message dans ses coefficients (par exemple, si le message binaire est 1011, on construit $x^3 + x + 1$).

— Pour faciliter la détection des bits à la déchiffre, on *multiplie* ce polynôme par $\lfloor q/2 \rfloor$. Dans Baby Kyber, cela revient à multiplier par 9 (car $\lfloor 17/2 \rfloor = 8$ ou 9, selon l'arrondi ; dans l'exemple présenté, on choisit 9).

$$\mathbf{m} = 9 \cdot \mathbf{m}_b.$$

— On choisit de nouveaux vecteurs de bruit (et de masquage) $\mathbf{r}$ et $\mathbf{e}_1, e_2$, également de « petits » coefficients.

— On calcule alors la paire $(\mathbf{u}, v)$:

$$\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1, \quad v = \mathbf{t}^T\mathbf{r} + e_2 + \mathbf{m}.$$

— Le *chiffre* (ciphertext) est le couple $(\mathbf{u}, v)$.

Déchiffrement Pour déchiffrer, le récepteur (qui possède $\mathbf{s}$) va :

— Calculer

$$\mathbf{m}_n = v \; - \; \mathbf{s}^T\mathbf{u},$$

— $\mathbf{m}_n$ n'est pas exactement $\mathbf{m}$, car il contient aussi un mélange de petits bruits. Mais les coefficients de $\mathbf{m}_n$ doivent être proches de 0 ou de $\lfloor q/2 \rfloor$.

— En « arrondissant » chaque coefficient vers 0 ou 9 (dans le cas de $q = 17$), on récupère le polynôme $\mathbf{m}$.

— Puis on divise par 9 pour obtenir $\mathbf{m}_b$, le polynôme binaire initial. Les coefficients (bits) retrouvés correspondent au message clair.

Pourquoi ce mécanisme fonctionne-t-il ? La clé réside dans le fait que :

$$v \; - \; \mathbf{s}^T\mathbf{u} \; \approx \; \mathbf{m} \; + \; (\text{petit bruit}).$$

Tant que ce « petit bruit » (lié aux coefficients $\mathbf{e}$ et $\mathbf{r}$ dans les polynômes) reste assez faible pour que le *seuil d'arrondi* distingue clairement les bits 0 et 1, le message est récupérable sans erreur.

Dans Baby Kyber, tout est réduit à de toutes petites dimensions (degré 4 des polynômes et $q = 17$). En pratique, Kyber *authentique* emploie des degrés nettement plus grands (ex. $n = 256$) et des moduli plus larges ($q = 3329$ dans les versions standardisées), ce qui rend bien sûr les instances MLWE beaucoup plus difficiles à attaquer.

En résumé, Baby Kyber illustre :

1. Comment une clé publique $(\mathbf{A}, \mathbf{t})$ est générée à partir d'une clé secrète $\mathbf{s}$ et d'un bruit $\mathbf{e}$.
2. Comment le chiffrement se fait en utilisant un masque $\mathbf{r}$ et des termes de bruit supplémentaires.
3. Comment le déchiffrement reconstitue la valeur du message à partir d'un « arrondi » des coefficients.

Ce principe est au cœur de Kyber (et, plus généralement, des schémas *LWE/MLWE-based*). Dans Kyber *réel*, on applique en plus des mécanismes de compression (pour réduire la taille des clés et des textes chiffrés) et des optimisations pour accélérer les calculs, tout en conservant le même *cœur* de procédé.

Points d'attention Comme pour tout schéma cryptographique, plusieurs précautions sont à prendre :

— **Implémentation constante en temps :** les opérations sur les polynômes et le bruit (générateurs de Gauss discret) doivent être effectuées de façon à éviter les *side-channels* (fuites temporelles, fuites d'information via la consommation électrique, etc.).

— **Génération aléatoire fiable :** la sécurité dépend grandement de la distribution de bruit. Un générateur aléatoire mal initialisé peut compromettre la robustesse face aux attaques.

— **Taille des clés et performances :** contrairement aux schémas classiques (Diffie-Hellman), la taille des clés et des échanges peut être plus importante, ce qui nécessite d'étudier l'impact sur les performances (coût en bande passante, stockage, etc.).

— **Usage *hybride* :** comme évoqué pour les signatures, l'utilisation conjointe d'un échange de clés classique (ECDH) et d'un échange basé sur les lattices (Kyber) reste souvent recommandée pour se prémunir d'éventuelles faiblesses nouvellement découvertes.

KyberSlash : fuites de secrets via des divisions dépendantes des données Bien que Kyber soit considéré comme un algorithme post-quantique solide, des travaux récents [7] ont mis en évidence une vulnérabilité baptisée *KyberSlash*. Cette vulnérabilité se manifeste dans de nombreuses implémentations de Kyber en raison d'une division dont le temps d'exécution varie selon la valeur du dividende, y compris lorsque celui-ci est « secret ». Plusieurs conditions rendent cette fuite d'informations potentiellement exploitable :

[7] Voir notamment [8].

1. **Division dépendante de la clé secrète :** un extrait de code présent dans les bibliothèques de référence effectue une division par la constante publique `KYBER_Q` (souvent 3329), mais le dividende (`t`) est issu d'un calcul dépendant de valeurs secrètes. Par exemple :

```
/* Extrait typique d'une implémentation de Kyber affectée */
void poly_tomsg(uint8_t msg[32], const poly *a)
{
  unsigned int i,j;
  uint16_t t;
  for(i=0;i<KYBER_N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t = a->coeffs[8*i+j];
      t += ((int16_t)t >> 15) & KYBER_Q;
      /* Division par KYBER_Q (publique), mais 't' est secret */
      t = (((t << 1) + KYBER_Q/2)/KYBER_Q) & 1;
      msg[i] |= t << j;
    }
  }
}
```

2. **Variation du temps de division :** dans de nombreux processeurs et environnements, l'instruction de division (ou la fonction d'« émulation » lorsqu'aucune instruction matérielle n'est disponible) a un temps d'exécution qui dépend des valeurs en entrée. Ainsi, si `t` atteint certains seuils (p. ex. 2048, 4096 ou 8192), le nombre de cycles peut changer, rendant le calcul chronométrable à distance.
3. **Situations concrètes de fuite :** des analyses empiriques (par exemple sur des CPU Zen 2 ou des plateformes RISC-V U74) ont montré des ralentissements ou accélérations de l'ordre de quelques cycles. Bien que ces variations puissent sembler faibles, il existe des techniques d'attaques par canaux auxiliaires suffisamment sensibles pour extraire des informations sur la clé secrète à partir d'une poignée de cycles d'écart.

KyberSlash1 et KyberSlash2. Les auteurs[8] ont répertorié deux grandes classes de divisions problématiques dans Kyber :

— **KyberSlash1** : la division apparaît durant la phase de *déchiffrement* (fonction `poly_tomsg`, par exemple), où la valeur `t` dépend directement des coefficients secrets.

[8] Voir la FAQ officielle KyberSlash [8] pour plus de détails.

— **KyberSlash2** : la division survient durant la phase de *chiffrement* (fonctions `poly_compress`, `polyvec_compress`, etc.), impliquant également des données secrètes.

Dans les deux cas, un attaquant capable de mesurer précisément les durées d'exécution peut « remonter » à des informations confidentielles, par exemple en proposant des messages chiffrés « spécialement conçus ».

Impacts et recommandations.

— **Environnements vulnérables :** toute plateforme où la division (ou la multiplication substituée par le compilateur) présente un temps d'exécution dépendant du dividende secret. Cela inclut des chaînes d'outils sur Intel/AMD (division matérielle non constante), ou sur ARM/RISC-V (division émulée via une fonction logicielle pouvant faire intervenir des branchements).

— **Portée de l'attaque :** certaines démonstrations pratiques, regroupées sous l'initiative *KyberSlash*, montrent la faisabilité d'extraire des secrets en moins d'une minute d'observation, dans certains cas et contextes. Même si l'utilisation unique d'une clé privée limite les attaques *multi-traces*, des travaux antérieurs suggèrent la prudence (il est déjà arrivé qu'on découvre ensuite des « single-trace attacks »).

— **Mitigations :** remplacer cette division dépendante des clés par des opérations à temps constant (par exemple, des multiplications et bitshifts correctement implémentés et indépendants des secrets). Divers « patches » et outils d'analyse (p. ex. `saferewrite`) sont disponibles pour détecter et corriger ces segments de code.

Origine du nom « KyberSlash ». Le terme *slash* fait référence à la barre oblique ("/") qui symbolise la division dans la plupart des langages de programmation et d'algèbre. Il ne faut pas confondre cette appellation avec une remise en cause de Kyber dans sa globalité : il s'agit spécifiquement d'une faiblesse des implémentations qui effectuent une division non protégée et dépendante de valeurs secrètes.

Conclusion sur les échanges de clés par lattices Tout comme les signatures à base de fonctions de hachage, les protocoles d'échange de clés par lattices illustrent à quel point la cryptographie post-quantique repose sur des idées connues (des produits scalaires, des masquages aléatoires, des algorithmes correcteurs d'erreurs, etc.) combinées de manière à résister aux attaques quantiques. Ces techniques ont déjà fait l'objet de nombreuses

analyses, dont certaines mettent en avant la nécessité de soigner chaque détail de l'implémentation.

À l'instar de *XMSS* ou *SLH-DSA* pour les signatures, des normes commencent à émerger (comme *CRYSTALS-Kyber* pour l'échange de clés), et il est probable qu'elles s'imposent dans de nombreuses infrastructures. Une fois encore, le fait de manipuler soi-même ces objets mathématiques (même avec un niveau de formalité modeste) permet d'acquérir une intuition précieuse et de démystifier la cryptographie post-quantique.

5 Conclusion

La cryptographie "post-quantique" n'est pas fondamentalement différente de la cryptographie dont nous avons tous l'habitude. Les quelques changements résident dans l'adaptation des tailles de clés et le choix de problèmes suffisamment robustes et pratiques pour résister aux attaques via des ordinateurs quantiques. En effet, avec une taille de clé publique d'1 téra-octet, même un algorithme comme RSA peut être considéré comme post-quantique [7] ! Cela ne serait toutefois pas très pratique à l'usage... En particulier, il est important de remarquer que les algorithmes développés pour la cryptographie "post-quantique" n'ont rien de quantique ; ils s'exécutent sur des ordinateurs classique. Ce sont les "attaquants" qui utilisent des ordinateurs quantique. Cela est à distinguer de la cryptographie quantique, avec par exemple l'Échange Quantique de Clé (QKD) [23], dont les algorithmes et les échanges se font sur des supports quantiques.

À travers la présentation du parcours de formation de l'équipe et du tour d'horizon du domaine présenté dans ce papier, nous espérons avoir démontré que la cryptographie post-quantique, au même titre que la cryptographie classique, est loin d'être inaccessible à un large public. En cybersécurité, les logiciels, les méthodologies, les environnements, les attaques et les pratiques évoluent constamment. De même qu'un cabinet d'audit ne réalise plus un test d'intrusion comme il le faisait 15 ans auparavant, le domaine de la cryptographie doit évoluer dans les prochaines années. Il est essentiel pour tous les acteurs du domaine de rester curieux et d'intégrer ces nouvelles connaissances à notre arsenal de compétences. Pour davantage de détails, l'audience est invitée à consulter le blog de Synacktiv, sur lequel plusieurs articles approfondis sur ce sujet ont déjà été publiés, et où Synacktiv continuera à partager ses recherches et analyses pour accompagner cette transition technologique importante.

Références

1. Multiplicative Learning With Errors and Cryptosystems. https://eprint.iacr.org/2011/119.
2. ANSSI. Avis de l'ANSSI sur la migration vers la cryptographie post-quantique. https://cyber.gouv.fr/publications/avis-de-lanssi-sur-la-migration-vers-la-cryptographie-post-quantique-0.
3. ANSSI. Mécanismes cryptographiques. https://cyber.gouv.fr/publications/mecanismes-cryptographiques.
4. ANSSI. PQC transition in France, ANSSI views. https://cyber.gouv.fr/sites/default/files/document/pqc-transition-in-france.pdf.
5. Asecuritysite. Quantum Robust : Hash to Obtain Random Subset (HORS). https://asecuritysite.com/signatures/hors.
6. Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, Aaron Kaiser, Peter Schwabe, Karolin Varner, and Bas Westerbaan. X-wing : The hybrid KEM you've been looking for. Cryptology ePrint Archive, Paper 2024/039, 2024.
7. Daniel J. Bernstein, Nadia Heninger, Paul Lou, and Luke Valenta. Post-quantum RSA. Cryptology ePrint Archive, Paper 2017/351, 2017.
8. S. Bhasin A. Chattopadhyay T. K. Chia M. J. Kannwischer F. Kiefer T. B. Paiva P. Ravi D. J. Bernstein, K. Bhargavan and G. Tamvada. KyberSlash : Exploiting secret-dependent division timings in Kyber implementations., 2025.
9. ETSI. ETSI TS 103 744 - CYBER - Quantum-safe Hybrid Key Exchanges. https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.01.01_60/ts_103744v010101p.pdf.
10. Gaëtan Ferry. Is it Post-Quantum time yet ? https://www.synacktiv.com/publications/is-it-post-quantum-time-yet.
11. Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. Cryptology ePrint Archive, Paper 2018/024, 2018.
12. Antoine Gicquel. Quantum Readiness : Hash-Based Signatures. https://www.synacktiv.com/publications/quantum-readiness-hash-based-signatures.
13. Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS : eXtended Merkle Signature Scheme. RFC 8391, May 2018.
14. NIST. FIPS 203 - Module-Lattice-Based Key-Encapsulation Mechanism Standard. https://csrc.nist.gov/pubs/fips/203/final.
15. NIST. FIPS 204 - Module-Lattice-Based Digital Signature Standard. https://csrc.nist.gov/pubs/fips/204/final.
16. NIST. FIPS 205 - Stateless Hash-Based Digital Signature Standard. https://csrc.nist.gov/pubs/fips/205/final.
17. NIST. IR 8545 - Status Report on the Fourth Round of the NIST Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/pubs/ir/8545/final.
18. NIST. Post-Quantum Cryptography. https://csrc.nist.gov/projects/post-quantum-cryptography.
19. NIST. SP 800-208 - Recommendation for Stateful Hash-Based Signature Schemes. https://csrc.nist.gov/pubs/sp/800/208/final.

20. Wikipédia. Algorithme BHT. https://en.wikipedia.org/wiki/BHT_algorithm.
21. Wikipédia. Apprentissage avec erreur. https://fr.wikipedia.org/wiki/Apprentissage_avec_erreurs.
22. Wikipédia. Arbre de Merkle. https://fr.wikipedia.org/wiki/Arbre_de_Merkle.
23. Wikipédia. Distribution quantique de clé. https://fr.wikipedia.org/wiki/Distribution_quantique_de_cl%C3%A9.
24. Wikipédia. Merkle Signature Scheme. https://en.wikipedia.org/wiki/Merkle_signature_scheme.
25. Wikipédia. Paradoxe des anniversaires. https://fr.wikipedia.org/wiki/Paradoxe_des_anniversaires.
26. Wikipédia. Ring Learning With Error. https://en.wikipedia.org/wiki/Ring_learning_with_errors.
27. Wikipédia. Réduction du paquet d'onde. https://fr.wikipedia.org/wiki/R%C3%A9duction_du_paquet_d%27onde.
28. C Zalka. Grover's Quantum Searching Algorithm is Optimal. https://arxiv.org/abs/quant-ph/9711070.