Blockchain implémentant un cryptosystème post-quantique

Louis Tremblay Thibault
<louis.tremblay.thibault@umontreal.ca>

27 avril 2022

1 Introduction

1.1 *Blockchain* et risques de sécurité

Les technologies de chaîne de blocs sont plus populaires que jamais. Les systèmes qui utilisent cette technologie permettent d'établir de manière sécuritaire et immutable des registres qui vivent sur des réseaux informatiques décentralisés. Ces systèmes se basent sur des idées du monde de la cryptographie pour assurer leur pérénnité. Cependant, la plupart des algorithmes de signature utilisés aujourd'hui dans les systèmes de chaîne de blocs sont basés sur le problème du logarithme discret [1] et sont donc vulnérables aux attaques proposées par les ordinateurs quantiques [11].

Comme les systèmes de chaîne de bloc sont couramment utilisés comme plateforme où les usagers peuvent transactionner et potentiellement échanger des actifs, il est primordial d'exiger une sécurité robuste lors du développement de ces systèmes. Néamoins, il semblerait que toutes les communautés actives à ce jour préfèrent utiliser un cryptosystème plus rapide mais vulnérable aux ordinateurs quantiques plutôt qu'un cryptosystème moins performant mais d'autant plus sécuritaire. Malgré le fait que les ordinateurs quantiques ne sont pas à ce jour assez puissants pour mener quelconque attaque, l'importance des enjeux entourant la sécurité des systèmes de chaîne de blocs impose d'être rigoureux et de minimiser les failles de sécurité possibles.

1.2 Travail proposé

Dans ce projet, nous introduisons un premier prototype de système de chaîne de bloc qui utilise un cryptosystème post-quantique. Ce cryptosystème (AMSS) est dit *hash based*, c'est-à-dire que sa seule dépendance est une fonction de hachage (par exemple SHA-256). AMSS fournit les primitives cryptographiques de base (génération de clés, signature, vérification de signature) de manière résistante aux possibles attaques des ordinateurs quantiques. Le prototype développé pour utiliser AMSS, $|coin\rangle$, est une implémentation simple d'un système de chaîne de blocs. Le mécanisme de consensus est *proof-of-work*, comme Bitcoin [10], et offre un système de compte/solde, comme Ethereum [3].

Nous étudierons en profondeur dans la section 2 le fonctionnement de la librairie AMSS, la section 3 offrira un survol du système $|coin\rangle$ et finalement la section 4 donnera une analyse du travail proposé.

2 AMSS

AMSS (A Merkle Signature Scheme) est une librairie qui a été développée dans le cadre du projet. Elle offre des primitives cryptographiques de base telles que la génération d'une paire de clés publique/privée, la signature d'un message et la validation d'une signature par rapport à une clé publique.

Les primitives offertes fonctionnent grâce à des techniques qui ne sont pas basées sur le problème du logarithme discret, mais bien sur les fonctions de hachage. De ce que nous connaissons des ordinateurs quantiques, un tel système est résistant aux attaques quantiques tandis qu'un système basé sur le problème du logarithme discret ne l'est pas.

Cette librairie a été construite avec des idées largement inspirées de [2] et [6].

2.1 Schéma de signature unique Winternitz *(W-OTS)*

Ce protocole de signature unique permet de générer une paire de clés publique/privée qui serviront à signer **un seul message une seule fois**. C'est la primitive de signature de notre cryptosystème post-quantique. Des enjeux de sécurité surviennent si plus d'une signature est publiée avec la même paire de clés, c'est pourquoi il est important d'utiliser chaque paire pour une seule signature.

W-OTS utilise une fonction de hachage $f : \{0,1\}^* \to \{0,1\}^n$. Dans notre implémentation, nous choisissons la fonction SHA256 que nous dénotons $H : \{0,1\}^* \to \{0,1\}^n$ où $n = 256$.

2.1.1 Génération des clés

Premièrement, un paramètre $w \geq 2$ est choisi. Ce paramètre représente le nombre de bits qui seront signés en même temps lors d'une signature. Sous la recommandation de [6], nous choisissons $w = 16$. Ensuite, le paramètre t est calculé selon les deux paramètres précédents suivant les équations suivantes

$$t_1 = \left\lceil \frac{n}{w} \right\rceil, \quad t_2 = \left\lfloor \frac{\lfloor \lg t_1 \rfloor + 1 + w}{w} \right\rfloor, \quad t = t_1 + t_2. \tag{1}$$

Dans notre cas, on obtient $t = 18$. On choisit ensuite aléatoirement t chaînes de bits de longueur $n = 256$, qui formeront notre clé de signature ou clé secrète :

$$S = (s_{t-1}, \ldots, s_1, s_0) \in \{0,1\}^{(n,t)}. \tag{2}$$

La clé publique P associée à la clé S est calculée en appliquant H $2^w - 1$ fois à chacune des t chaînes de bits de S :

$$P = (p_{t-1}, \ldots, p_1, p_0) \in \{0,1\}^{(n,t)} \tag{3}$$

où $p_i = H^{2^w-1}(s_i)$.

2.1.2 Génération de signature

Pour un message donné M, on calcule $d = H(M)$ qu'on décompose en $t_1 = 16$ chaînes de bits de longueur w :

$$d = b_{t-1}||b_{t-2}||\ldots||b_{t-t_1} \tag{4}$$

où || dénote la concaténation de chaînes de bits. Ensuite, on calcule la somme de contrôle suivante :

$$c = \sum_{i=t-t_1}^{t-1} (2^w - (b_i)_{10}) \tag{5}$$

où $(b_i)_{10}$ représente la valeur en base 10 de la chaîne de bits b_i, qui sera bien sûr comprise entre 0 et $2^w - 1$.

On ajoute au besoin un nombre de bits 0 à gauche de la représentation binaire de c pour que la longueur de celle-ci soit divisible par w, et on séparera la chaîne de bits résultante en $t_2 = 2$ chaînes de w bits :

$$c = b_{t_2-1}||b_0. \tag{6}$$

Maintenant que nous avons t chaînes de bits b_i, nous pouvons calculer la signature du message

$$\sigma = (\sigma_{t-1}, \ldots, \sigma_1, \sigma_0) \in \{0,1\}^{(n,t)} \tag{7}$$

où $n_i = (b_i)_{10}$ et $\sigma_i = H^{n_i}(s_i)$.

2.1.3 Vérification de signature

Étant donné une signature $\sigma = (\sigma_{t-1}, \ldots, \sigma_1, \sigma_0) \in \{0,1\}^{(n,t)}$ et un message M, l'agent vérificateur calcule $d = H(M)$ et les chaînes de bits $b_{t-1}, \ldots, b_1, b_0$ comme dans la section précédente. Ensuite, l'agent vérifie si

$$(H^{2^w-1-n_{t-1}}(\sigma_{t-1}), \ldots, H^{2^w-1-n_1}(\sigma_1), H^{2^w-1-n_0}(\sigma_0)) = (p_{t-1}, \ldots, p_1, p_0). \tag{8}$$

Si la signature est valide, on a

$$\begin{aligned} \sigma_i = H^{n_i}(s_i) \implies H^{2^w-1-n_i}(\sigma_i) &= H^{2^w-1-n_i}(H^{n_i}(s_i)) \\ &= H^{2^w-1}(s(i)) \\ &= p_i. \end{aligned}$$

On a donc que si l'égalité (8) tient, la signature est valide.

2.2 Schéma de signature de Merkle MSS

Le système W-OTS décrit précédemment nous permet de générer une seule signature. L'idée derrière le schéma de signature de Merkle est de prégénérer un nombre arbitrairement grand de paires de clés W-OTS et de regrouper toutes les clés publiques sous une seule en utilisant un arbre de Merkle. La racine de cet arbre agit donc comme la clé publique du système. La clé secrète du système MSS est la séquence des clés secrètes W-OTS.

Il est à noter que dans ce système, le nombre de signatures qu'un agent peut apposer est arbitrairement grand mais **fini**, ce qui le différencie des autres systèmes largement utilisés aujoud'hui, comme ECDSA ou RSA. En contrepartie, MSS n'est pas présentement vulnérable aux attaques quantiques tandis que les autres systèmes cités le sont.

Le système MSS requiert lui aussi l'utilisation d'une fonction de hachage. Nous choisissons une fois de plus la fonction SHA256 dénotée $H : \{0,1\}^* \to \{0,1\}^n$ avec $n = 256$.

2.2.1 Génération des clés

Premièrement, le paramètre h, représentant la hauteur de l'arbre de signature, est choisi. Pour un h donné, on pourra signer au plus 2^h messages puisque chacune des feuilles de l'arbre sera associée à une paire W-OTS. Autrement dit, on aura droit à une signature par feuille de l'arbre binaire plein de hauteur h. Ensuite, on génère 2^h paires de clés W-OTS comme vu à la section 2.1.1. Nous obtenons donc les paires de clés

$$(S_i, P_i),\ \forall i \in \{0, \ldots, 2^h - 1\}. \tag{9}$$

On attribue à chacune des $2^h - 1$ feuilles de l'arbre le résultat du hachage de la clé publique W-OTS P_i correspondante. Autrement dit, on a

$$\nu_{h,i} = H(P_i),\ \forall i \in \{0, \ldots, 2^h - 1\} \tag{10}$$

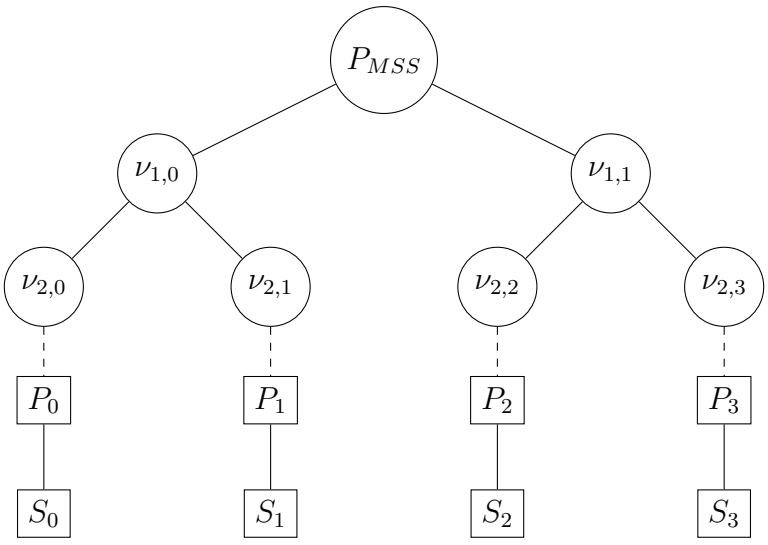
où $\nu_{\eta,i}$ désigne le i-ème nœud à la hauteur η.

Dans un arbre de Merkle, les nœuds internes prennent la valeur du résultat du hachage de la concaténation des deux nœuds enfants. Plus formellement, on a

$$\nu_{\eta,i} = H(\nu_{\eta+1,2i} || \nu_{\eta+1,2i+1}), \quad \eta \in \{0, \ldots, h-1\}, i \in \{0, \ldots, 2^\eta - 1\}. \tag{11}$$

L'arbre de signature MSS est maintenant généré. Comme mentionné plus haut, la clé publique de ce système est la valeur que prend la racine de l'arbre, qui est dénotée P_{MSS}. La figure 1 montre un exemple d'un arbre MSS généré avec $h = 4$.

FIGURE 1 – Un arbre de signature de Merkle avec $h = 2$



2.2.2 Génération de signature

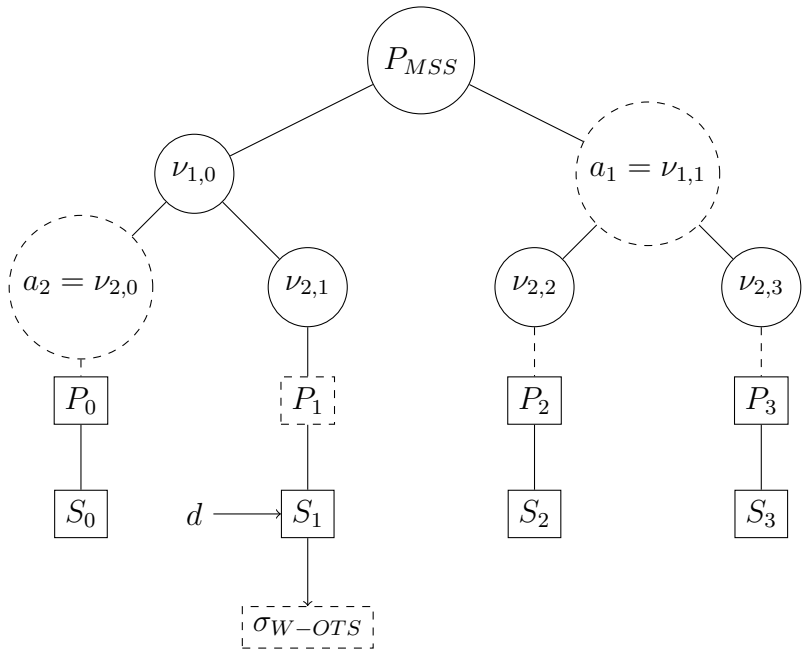
La i-ème signature générée par le système MSS σ_i contient, entre autres, la signature générée par la i-ème clé secrète S_i. Autrement dit, σ_i contient la signature du système W-OTS σ_{W-OTS} obtenue avec la clé S_i, la clé publique associée P_i et la valeur de l'indice i. Pour montrer que P_i est bien une des clés associées à l'arbre de signature de racine P_{MSS}, on inclut dans la signature les h

nœuds de l'arbre qui permettront au vérificateur de calculer la racine et la comparer à P_{MSS}. Ces nœuds sont appelés une preuve de Merkle. Une telle preuve montre qu'un certain bloc de données (dans ce cas-ci une clé publique P_i) appartient bien à un arbre de Merkle (dans ce cas-ci celui dont la racine est P_{MSS}).

Alors, la signature d'un message dans notre schéma proposé peut être définie comme un ensemble d'objets $\sigma_i = \{\sigma_{W-OTS}, P_i, i, \{a_1, \ldots, a_h\}\}$.

La figure 2 montre un exemple des données à inclure dans la signature pour qu'elle soit vérifiable et que la preuve de Merkle soit complète.

FIGURE 2 – Construction d'une signature MSS avec $i = 1$



2.2.3 Vérification de signature

Étant donné une signature $\sigma_i = \{\sigma_{W-OTS}, P_i, i, \{a_1, \ldots, a_h\}\}$, on peut retrouver la clé publique du signataire (comme on le ferait dans un schéma comme ECDSA).

Premièrement, un vérificateur vérifie la signature σ_{W-OTS} comme décrit à la section 2.1.3. Si cette vérification échoue, on discarte complètement la signature σ_i et le processus est terminé.

Si par contre la signature est valide, on peut poursuivre en vérifiant la preuve de Merkle à chaque niveau de l'arbre de la manière suivante :

$$\pi_\eta = \begin{cases} H(\pi_{\eta+1}||a_{\eta+1}) \text{ si } \left\lfloor \frac{i}{2^{h-\eta-1}} \right\rfloor \equiv 0 \pmod 2 \\ H(a_{\eta+1}||\pi_{\eta+1}) \text{ si } \left\lfloor \frac{i}{2^{h-\eta-1}} \right\rfloor \equiv 1 \pmod 2 \end{cases} \tag{12}$$

où $\eta \in \{0, \ldots, h-1\}$, $\pi_h = H(P_i)$ et $||$ dénote la concaténation. Par construction, la dernière itération donnera $\pi_0 = P_{MSS}$. Nous avons donc retrouvé la clé publique du système qui a généré la signature σ_i. Pour authentifier le signataire, on peut aussi comparer la clé publique associée à la signature avec la clé publique fournie par le signataire.

3 *ketcoin*

$|coin\rangle$ utilise une implémentation maison de AMSS pour obtenir les primitives cryptographiques de génération de clé, signature de message et vérification de signature qui sont résistantes aux attaques proposées par les ordinateurs quantiques. Il s'agit d'une innovation dans le monde des chaînes de bloc. En effet, comme les autres systèmes cherchent la rapidité à tout prix, ils évitent d'utiliser des cryptosystèmes un peu plus lents mais plus sécuritaires face aux menaces futures.

Comme il ne s'agit que d'un prototype, $|coin\rangle$ met en place un mécanisme de consensus *proof-of-work* sans ajustement de difficulté. $|coin\rangle$ n'utilise pas de DAG ou de UTXO, mais bien un système de compte et solde.

Tout comme la librairie AMSS, $|coin\rangle$ et été développé en langage Go et en ne faisant appel qu'aux librairies standard.

Le réseau est présentement en ligne et l'on peut s'y connecter avec le logiciel téléchargeable `https://github.com/tremblaythibaultl/ketcoin`. Les instructions pour se connecter au réseau sont disponibles dans le fichier `README.md`

4 Analyse

La section suivante analyse la performance du système AMSS en supposant une approche naïve pour la générations des clés et la vérification des signatures. Plusieurs formes d'optimisation pourraient être intégrées au système lors de travaux futurs.

Le présent tableau se base sur les paramètres w, h et n qui sont choisis par l'usager. Dans notre implémentation, nous avons $w = 16$, $h = 8$ et $n = 256$. Les temps sont calculés en évaluations de H et les tailles en bits.

	ECDSA (secp256r1)	AMSS
Temps de génération de clé	0	$2^h \cdot t \cdot (2^w - 1) + 2^{h+1} - 1$
Taille de la clé privée	256	$2^{h+1} \cdot t \cdot n + n \cdot (2^{h+1} - 1) + h$
Taille de la clé publique	≈ 257	n
Temps de génération de signature	0	$\leq t \cdot (2^w - 1)$
Taille de la signature	512	$2 \cdot (t \cdot n) + h \cdot n + h$
Temps de vérification de signature	0	$\leq t \cdot (2^w - 1) + h$

Tandis que la taille d'une clé dans un cryptosystème couramment utilisé comme ECDSA est entre 163 et 571 bits [9] (256 sur la courbe secp256k1), la taille de la clé publique dans AMSS est fixée à 256 bits et la taille de la clé privée est nettement plus grande, soit un peu plus de 1 MB pour une capacité de 1024 signatures. Il s'agit là d'un réel désavantage par rapport aux méthodes traditionnelles, qui offrent une quantité illimitée de signatures pour une paire de clés donnée. De plus, avec la taille de la clé publique vient un temps de génération de clé initial non négligeable associé à un nombre d'évaluation de la fonction de hachage dans $O(2^{h+1})$. En comparaison, ECDSA offre un temps de génération de clé qui est négligeable.

Pour ce qui est des signatures, celles des systèmes couramment utilisés ont une taille de 512 bits [8]. Un système AMSS à 1024 signatures offre des signatures de taille ≈ 1 kB.

En termes de temps, ECDSA ne requiert pas d'évaluation de fonction de hachage. Cependant, les opérations mathématiques complexes nécessaires au bon fonctionnement de ce système amènent tout de même un temps non nul [7].

Outre la complexité, le tableau suivant met en lumière les différences de temps d'éxécution des primitives de signature sur un ordinateur muni d'un processeur Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz pour un système AMSS permettant de signer 1024 messages ($h = 10$, $n = 256$, $w = 16$, $t = 18$). Les temps sont calculés en secondes.

	ECDSA (secp256r1)	AMSS
Temps de génération de clé	$\approx 2,5128 \times 10^{-5}$	$\approx 397,5089$
Temps de génération de signature	$\approx 2,10596 \times 10^{-4}$	$\approx 0,1374$
Temps de vérification de signature	$\approx 1,51299 \times 10^{-4}$	$\approx 0,2363$

De manière générale, les performances de notre système sont raisonnables mais nettement moins bonnes que les performances des système standards d'aujourd'hui. Par contre, ces systèmes sont vulnérables aux attaques portées par ordinateur quantique tandis que AMSS ne l'est pas.

Les cryptosystèmes *hash-based*, comme AMSS, sont considérés résistants aux attaques menées par les ordinateurs quantiques parce qu'il n'existe pas d'algorithme quantique efficace pour trouver x étant donné $y = H(x)$. En effet, les seuls algorithmes quantiques connus peuvent trouver une collision dans une fonction de hachage cryptographique en temps $O(2^{n/2})$ plutôt qu'en temps $O(2^n)$ avec un ordinateur classique [4]. La communauté scientifique est d'avis que doubler la taille de sortie de la fonction de hachage (utliser SHA-512 au lieu de SHA-256) contrera la rapidité quantique proposée.

5 Conclusion

Nos travaux innovent dans l'espace des systèmes de chaîne de blocs en proposant un premier prototype combinant chaîne de blocs décentralisée et cryptographie post-quantique. La librairie AMSS offre des primitives cryptographiques de manière résistante aux attaques proposées par les algorithmes quantiques et le prototype $|coin\rangle$ implémente un système de chaîne de blocs simple qui utilise la librairie AMSS comme primitive cryptographique. Offrant une sécurité cryptographique avancée, il s'agit d'un bon point de départ sur lequel la communauté pourra s'appuyer.

Cependant, nos systèmes ont pour l'instant des perfomances plutôt faibles. Plusieurs améliorations simples sont réalisables et la section suivante en énumère quelques unes.

5.1 Travaux futurs

Voici quelques propositions d'améliorations pour les systèmes AMSS et $|coin\rangle$.

— Utiliser W-OTS$^+$ [5] comme primitive de signature unique au lieu de W-OTS.

— Utiliser un générateur de nombres pseudo-aléatoires pour la génération des clés MSS.

— Utiliser un meilleur algorithme de traverse d'arbre. Cela rendrait la sauvegarde et conservation des clés MSS plus efficace.

— Implémenter un mécanisme de consensus *proof-of-stake* dans $|coin\rangle$.

— Raffiner le code de $|coin\rangle$,le rendre *thread-safe* et gérer toutes les exceptions.

Ces propositions permettront aux systèmes d'obtenir de meilleures performances autant en stockage et transmission de données qu'en temps de calcul.

Références

[1] Robert Annessi and Ethan Fast. Improving security for users of decentralized exchanges through multiparty computation. In *2021 IEEE International Conference on Blockchain (Blockchain)*, pages 229–236. IEEE, 2021.

[2] Johannes Buchmann, Erik Dahmen, and Michael Szydlo. *Hash-based Digital Signature Schemes*, pages 35–93. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[3] Vitalik Buterin et al. Ethereum white paper. *GitHub repository*, 1 :22–23, 2013.

[4] André Chailloux, María Naya-Plasencia, and André Schrottenloher. An efficient quantum collision search algorithm and implications on symmetric cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 211–240. Springer, 2017.

[5] Andreas Hülsing. W-ots+ – shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *Progress in Cryptology – AFRICACRYPT 2013*, pages 173–188, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[6] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. Xmss : extended merkle signature scheme. In *RFC 8391*. IRTF, 2018.

[7] Nicholas Jansma and Brandon Arrendondo. Performance comparison of elliptic curve and rsa digital signatures. *nicj. net/files*, 2004.

[8] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1) :36–63, 2001.

[9] Sharon Levy. Performance and security of ecdsa. *Computer Science*, 2015.

[10] Satoshi Nakamoto. Bitcoin whitepaper. *URL : https ://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*, 2008.

[11] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *arXiv preprint quant-ph/0301141*, 2003.