

Quantum Resistant Ledger (QRL)

peterwaterland@gmail.com

November 2016

Abstract

Private, digitale Währungen müssen insbesondere gegenüber den Fortschritten der modernen IT sicher sein, um eine Langlebigkeit zu erreichen. Präsentiert werden Design und Erstellung eines Kryptowährungs-ledgers unter Verwendung von Hash-basierten, digitalen Signaturen, das sowohl gegenüber herkömmlichen Angriffen als auch gegenüber Quantencomputer-Attacken resistent sind.

1 Einleitung

Über das Konzept eines Peer-to-Peer Internet-Ledger im Rahmen einer Blockchain, das mittels eines Ausführungs-nachweises abgesichert wird, wurde zum ersten Mal im Jahr 2008 berichtet[11]. Bitcoin ist bis heute am häufigsten genutzte Kryptowährung. Hunderte von ähnlichen Kryptoledgerwährungen wurden seitdem entwickelt, jedoch mit Ausnahme weniger Währungen greifen diese auf dieselbe Elliptic Curve Cryptography (ECDSA) zurück, um digitale Signaturen zu erzeugen, mit denen die Transaktionen sicher verifiziert werden können. Die am häufigsten genutzten Signaturverfahren wie ECDSA, DAS und RSA sind theoretisch anfällig gegenüber Quantencomputer-Attacken. Somit erscheint es als sinnvoll, das Design und die Konstruktionsmerkmale eines quantenresistenten Blockchain-Ledgers zu erkunden, um dem möglichen plötzlichen, nicht linearen Fortschritt in der Quantencomputertechnik entgegentreten zu können.

2 Bitcoin Transaktionssicherheit

Es ist zurzeit lediglich möglich, von einer Bitcoin-Adresse aus, Transaktionen zu tätigen (noch nicht ausgegebener Transaktionsoutput), indem eine Transaktion mit einer validen elliptischen Kurvensignatur (secp256k1) aus dem privaten Key ($x \in N | x < 2^{256}$) der spezifischen Bitcoin-Adresse signiert wird. Wenn der tatsächlich zufällig generierte private Schlüssel geheim gehalten wird oder verloren geht, ist nicht zu erwarten, dass jemals Geldmittel von dieser Adresse aus transferiert werden können. Die Wahrscheinlichkeit einer effektiven Übereinstimmung zweier privater Bitcoin-Private-Keys liegt bei 1 zu 2256. Die Wahrscheinlichkeit der Übereinstimmung einer jeden Bitcoin-Adresse kann mithilfe des Geburtstagsparadoxons ermittelt werden. Die Anzahl der Bitcoin-Adressen, die generiert werden müssen, um die 0.1% Chance einer Übereinstimmung zu erreichen, liegt bei 5.4×10^{23} [14].

Unabhängig davon wird bei der Signierung einer Transaktion der öffentliche ECDSA-Schlüssel der sendenden Adresse offengelegt und in der Blockchain gespeichert. Die beste Herangehensweise ist es, Adressen nicht wiederzuverwenden - jedoch sind derzeit (Stand November 2016) 49.56% der gesamten Bitcoin-Guthaben in Adressen mit frei zugänglichen öffentlichen Schlüsseln gelagert [1].

3 Angriffsvektoren Quantencomputing

Aufgrund hoher rechnerischer Anforderungen bei der Faktorisierung von großen Zahlen, dem diskreten Logarithmusproblem und dem Problem des diskreten Logarithmus von elliptischen Kurven können die Verfahren RSA, DSA und ECDSA als sicher bezeichnet werden. Shors Quantenalgorithmen (1994) löst das Problem der Faktorisierung von großen Zahlen und von diskreten Logarithmen in

polynomischen Zeitalgorithmen. Daher kann ein Quantencomputer theoretisch den privaten Schlüssel eines gegebenen öffentlichen ECDSA Schlüssels rekonstruieren. Es wird angenommen, dass ECDSA durch die Verwendung kleinerer Schlüssellängen verwundbarer gegenüber Quantencomputer-Attacken ist als RSA – da ein 1300 und 1600 Qubit (211) Quantencomputer in der Lage ist, ein 228 Bit ECDSA zu errechnen.

Die Entwicklung öffentlich zugänglicher Quantencomputer hat bisher die Marke von 25 Qubits oder die Faktorisierung von kleinen Zahlen (15 oder 21) noch nicht überschritten. Die NSA missbilligte im August 2015 die elliptische Kurven-Kryptographie, die scheinbar auf der Quantencomputer-Technik basiert. Es ist unklar, wie weit fortgeschritten die Entwicklung eines Quantencomputers ist. Zudem ist unklar, ob Durchbrüche auf diesem Gebiet überhaupt publiziert werden, um allgemein im Gebrauch befindliche kryptographische Protokolle in der Post-Quantum-Ära noch nutzbar zu halten. Aufgrund seiner Ursprünge im Anti-Establishment könnte die Bitcoin-Währung eines der ersten Ziele von Gegnern der Quantencomputer darstellen.

Würde eine signifikante Weiterentwicklung der Quantencomputer stattfinden, könnten Entwickler quantenresistente, kryptographische Signaturschemata in Bitcoin implementieren und alle Benutzer dazu ermutigen, ihre Guthaben von ECDSA-basierten Adressen auf neue, quantenresistente Adressen zu transferieren. Um den Anteil der betroffenen Adressen möglichst gering zu halten, wäre es sinnvoll, die Wiederverwendung von öffentlichen Schlüsseln auf der Protokollbene zu deaktivieren. Ein solch geplantes Upgrade würde wahrscheinlich auch zu einer Bewegung der 1 Millionen Coins, die sich im Besitz von Satoshi Nakamoto befinden, führen und damit einhergehende Preisschwankungen auslösen.

Ein weniger günstiges Szenario wäre ein stiller, nichtlinearer Fortschritt im Quantencomputing, gefolgt von einer differenzierten Quantencomputer-Attacke auf Bitcoin-Adressen mit bekannten öffentlichen Schlüsseln. Solche Diebstähle könnten, bedingt durch den großen Verkaufsdruck und den kompletten Verlust des Vertrauens in das System, einen verheerenden Effekt auf den Handelspreis von Bitcoin haben, wenn das Ausmaß dieser Attacken bekannt würde. Das Vertrauen in Bitcoin als Wertanlage ('digitales Gold') würde sehr schwer beschädigt werden - mit extremen Konsequenzen für die ganze Welt. Vor diesem Hintergrund halten es die Autoren für sinnvoll, mit quantenresistenten kryptographischen Signaturen zu experimentieren sowie möglicherweise eine Backup-Wertanlage für den Fall eines Schwarzen Schwans in petto zu haben.

4 Quantenresistente Signaturen

Es existieren einige wichtige kryptographische Systeme, von denen angenommen wird, dass sie quantenresistent sind: Hash-basierte Kryptographie, Code-basierte Kryptographie, Gitter-basierte Kryptographie, multivariate-quadratische Kryptographie-Systeme und Kryptographie unter Zuhilfenahme privater Schlüssel. Von all diesen Schemata wird angenommen, dass sie sowohl klassischen als auch quantenbasierten Rechenangriffen widerstehen können, sobald ausreichend lange Schlüssel verwendet werden.

Zukunftssichere hash-basierte digitale Signaturschemata existieren mit minimalen Sicherheitsanforderungen, welche sich ausschließlich auf die Kollisionsresistenz einer kryptografischen Hash-Funktion verlassen. Verändert man die gewählte Hash-Funktion, erzeugt dies ein neues hash-basiertes digitales Signaturschema. Hash-basierte digitale Signaturen sind gut erforscht und stellen die Post-Quantum-Signaturen der Zukunft dar. Als solche entsprechen sie der auserwählten Klasse für Post-Quantum-Signaturen für das QRL.

5 Hash-basierte digitale Signaturen

Quantenresistente hash-basierte Signaturen beruhen auf der Sicherheit einer einseitigen kryptographischen Hash-Funktion, die eine Nachricht m annimmt und einen Hash-Digest h der festgelegten Länge n z.B. SHA-256, SHA-512. Unter Verwendung einer kryptographischen

Hash-Funktion sollte es rechnerisch unmöglich sein, m von h (Pre-Image resistant) zu errechnen oder h von h_2 zu errechnen, solange $h_2 = \text{hash}(h)$ (Zweite Pre-Image-Resistenz) ist, wobei es sehr schwer ist, zwei Nachrichten ($m_1 \neq m_2$) zu finden, die dasselbe h erzeugen (Kollisionsresistenz). Grovers Quantenalgorithmus kann verwendet werden, um eine Hash-Kollision zu finden oder einen Pre-Image-Angriff durchzuführen, um m zu finden, was $O(2^{n/2})$ Rechenvorgänge erfordert. Um also eine 128-Bit-Sicherheit zu erhalten, muss also eine Hash-Digest-Länge n von mindestens 256-Bit ausgewählt werden – eine perfekte kryptographische Hash-Funktion vorausgesetzt.

Hash-basierte Signaturen erfordern einen öffentlichen Schlüssel pk zur Verifizierung und einen privaten Schlüssel sk zur Unterzeichnung der Nachricht. Verschiedene hash-basierte Einmalsignaturen (OTS) werden bezüglich ihrer Eignung für die Verwendung als Teil eines Blockchain-Ledgers diskutiert.

5.1 Lamport-Diffie OTS

1979 beschrieb Lamport eine hash-basierte Einmalsignatur für eine Nachricht mit der Länge m Bits (üblicherweise die Ausgabe einer kollisionsresistenten Hashfunktion). Die Generierung eines Schlüsselpaares erzeugt m Paare von zufälligen privaten Schlüsseln $sk_j^m \in \{0, 1\}^n$ wobei $j \in \{0, 1\}$ ist z.B. der private Key $sk = ((sk_0^1, sk_1^1), \dots, (sk_0^m, sk_1^m))$. Wenn f eine Ein-Weg-Hashfunktion $\{0, 1\}^n \rightarrow \{0, 1\}^n$ mit m Paaren von generierten öffentlichen Schlüsseln $pk_j^m = f(sk_j^m)$ ist beispielsweise der öffentliche Schlüssel $pk = (pk_0^1, pk_1^1), \dots, (pk_0^m, pk_1^m)$. Das Unterschreiben beinhaltet die bitweise Überprüfung des Nachrichtenhashs, um sk_j auszuwählen, (z.B. wenn $bit = 0, sk_j = sk_0$, $bit = 1, sk_j = sk_1$), was zur Erstellung der Signatur $s = (sk_0^1, \dots, sk_j^m)$ führt. Dies legt den halben, privaten Schlüssel offen. Um die Signatur zu verifizieren, stellt die bitweise ($j \in \{0, 1\}$) Überprüfung des Nachrichtenhashs sicher, dass $(pk_j = f(sk_j))_m$ entspricht.

Nimmt man an, dass nach Grovers Algorithmus 128 Bit Sicherheit gewünscht wird, wobei die Nachrichtenlänge ein Ergebnis des festgelegten Hashs von SHA256 $m = 256$ und $n = 256$ ist, ist das Resultat $pk = sk = 16\text{kb}$. Es entsteht eine Signatur von je 8kb für jeden OTS, der benutzt wird. Eine Lamport-Signatur sollte nur einmal benutzt werden. Sie kann zwar sehr schnell generiert werden, leidet jedoch unter den großen Schlüsseln, Signaturen und erweiterten Transaktionsgrößen. Dies macht ihre Verwendung für öffentliche Blockchain-Ledger unzweckmäßig.

5.2 Winternitz OTS

Für einen Message-Digest M mit der Länge m Bits, einem privaten und öffentlichen Schlüssel der Länge n Bits, einer Einweg-Funktion $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ und einem Winternitz-Parameter $w \in N | w > 1$ besteht die generelle Idee der Winternitz Einmalsignatur darin, eine wiederholende Hash-Funktion auf eine Liste von zufällig erzeugten privaten Schlüsseln, $sk \in \{0, 1\}^n$, $sk = (sk_1, \dots, sk_{m/w})$ anzuwenden, welche eine Kette von Hashes in der Länge $w - 1$ erzeugt und mit den öffentlichen Schlüsseln $(pk \in N | \{0, 1\}^n)$, $pk_x = f^{2^{w-1}}(sk_x)$, $pk = (pk_1, \dots, pk_{m/w})$ endet.

Anders als die bitweise Überprüfung des Message-Digest in der Lamport Signatur, wird stattdessen die Nachricht mit w Bits zur selben Zeit geparst, um die Nummer $i \in N, i < 2^w - 1$ zu erhalten, aus der die Signatur $s_x = f^i(sk_x)$, $s = (s_1, \dots, s_{m/w})$ generiert wird. Ein steigendes w bietet hierbei einen Konflikt zwischen kürzeren Schlüsseln und Signaturen für einen erhöhten rechnerischen Aufwand [10]. Die Verifikation besteht aus einer einfachen Erzeugung von $pk_x = f^{2^{w-1}-i}(s_x)$ von M , s und der Bestätigung, dass die öffentlichen Schlüssel übereinstimmen.

Die Verwendung von SHA-2 (SHA-256) als Einweg-Kryptographie Hash-Funktion, f : $m = 256$ und $n = 256$, mit $w = 8$, resultiert in einer $pk = sk = s$ Größe von $\frac{(m/w)n}{8}$ bytes = 1kb. Um pk zu generieren wird eine f^i Hashwiederholung benötigt, wobei $i = \frac{m}{w}2^{w-1} = 8160$ je OTS-Schlüsselgeneration entspricht. Wenn $w = 16$, sind die Schlüssel und Signaturen halb so groß, bei $i = 1048560$ wird dieser Prozess impraktikabel.

5.3 Winternitz-OTS

Buchmann stellte eine Variante des ursprünglichen Winternitz OTS vor, indem diese eine iterierende Einweg-Funktion verändert, sodass diese stattdessen auf eine zufällige Zahl x wiederholt angewendet werden kann – diesmal jedoch parametrisiert mit einem Schlüssel, k , der von der vorherigen Iteration von $f_k(x)$ generiert wurde. Diese ist bei adaptiv gewählten Mitteilungsattacken nahezu fälschungssicher, wenn eine pseudo-zufällige Funktion (PRF, pseudo random function) genutzt wird und ein Sicherheitsnachweis für die gegebenen Parameter errechnet werden kann[3]. Dies eliminiert die Notwendigkeit einer kollisionsresistenten Hash-Funktionsfamilie durch die Durchführung einer zufälligen Überprüfung statt der simplen Iteration. Huelsing stellte eine weitere Variante des W-OTS+ vor, welche die Erstellung kleinerer Signaturen unter Beibehaltung einer äquivalenten Bit-Sicherheit durch die Addition einer Bitmaske XOR in der iterativen Verkettungsfunktion ermöglicht[6]. Ein weiterer Unterschied zwischen W-OTS (Variante 2011) / W-OTS+ und W-OTS ist, dass die Nachricht zu einem Zeitpunkt $\log_2(w)$ Bits statt bei w geparsst wird, was zu geringeren Iterationen führt, die Schlüssellänge und Signaturgröße jedoch vergrößert.

W-OTS+ soll nun kurz beschrieben werden. Mit den Sicherheitsparametern $n \in N$, die der Nachrichtenlänge der Nachricht (m) entsprechen, werden die Schlüssel und Signaturen in Bits durch die jeweilig gewählte kryptographische Hash-Funktion und den Winteritz-Parameter $w \in N | w > 1$ (normalerweise {4,16}) bestimmt. l wird errechnet. l ist eine Nummer von n bit Stringelementen in einem WOTS+ Schlüssel oder einer Signatur, wobei $l = l_1 + l_2$:

$$l_1 = \lceil \frac{m}{\log_2(w)} \rceil, \quad l_2 = \lfloor \frac{\log_2(l_1(w-1))}{\log_2(w)} \rfloor + 1$$

Eine Keyed-Hash-Funktion wird verwendet

$$f_k : \{0, 1\}^n \rightarrow \{0, 1\}^n \mid k \in \{0, 1\}^n$$

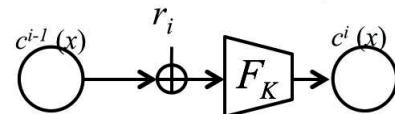
In Pseudocode:

$$f_k(M) = \text{Hash}(Pad(K) || Pad(M))$$

wobei $Pad(x) = (x || 10^{b|x|})$ für $|x| < b$.

Die Verkettungsfunktion $c_k^i(x, r)$: basierend auf der Eingabe $x \in \{0, 1\}_n$ der Iterationszähler i der Schlüssel $k \in K$ und die Zufälligkeitselemente $r = (r_1, \dots, r_j) \in \{0, 1\}^{n * j}$ mit $j \geq i$ wird wie folgt definiert:

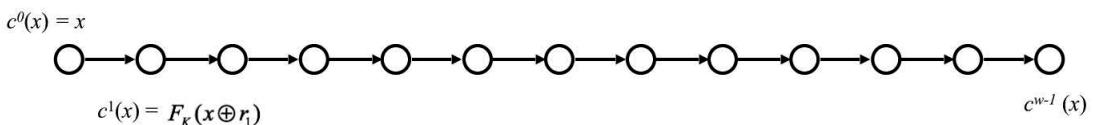
Abbildung 1. W-OTS+ Verkettungsfunktion



wobei

$$c^i(x, r) = \begin{cases} x & \text{if } i = 0; \\ f_k(c_k^{i-1}(x, r) \oplus r_i) & \text{if } i > 0; \end{cases}$$

Abbildung 2. Beispielgenerierung einer Hash-Kette



Dies ist eine bitweise XOR-Operation der vorhergehenden Iteration von c_k und dem Zufälligkeits-element, gefolgt vom Ergebnis von f_k welches dann der nächsten Iteration von c_k zugeführt wird.

5.3.1 Signaturschlüssel

Um einen geheimen Schlüssel sk zu generieren, werden, $l+w-1$ n bit Strings (mit PRF) gleichverteilt zufällig ausgewählt, aus denen der geheime Schlüssel $sk = (sk_1, \dots, sk_l)$ generiert wird. Die verbleibenden $w-1$ n bit Strings werden zu $r = (r_1, \dots, r_{w-1})$. Ein Funktionsschlüssel k wird gleichverteilt zufällig erzeugt.

5.3.2 Prüfschlüssel

Der öffentliche Schlüssel ist:

$$pk = (pk_0, pk_1, \dots, pk_l) = ((r, k), c_k^{w-1}(sk_1, r), c_k^{w-1}(sk_2, r), \dots, c_k^{w-1}(sk_l, r))$$

Hinweis: pk_0 enthält r und k .

5.3.3 Unterscheiben

Die Signatur wird wie folgt erzeugt: Die Nachricht M , mit der Länge m , wird geparsst, sodass $M = (M_1, \dots, M_l)$, $M_i \in \{0, w-1\}$ (dies erzeugt eine w -Basis, die M repräsentiert). Als nächstes wird die Prüfsumme C berechnet, welche die Länge l_2 hat, und wird wie folgt angehängt:

$$C = \sum_{i=1}^{l_1} (w - 1 - M_i)$$

Sodass: $M + C = b = (b_0, \dots, b_l)$

Die Signatur ist:

$$s = (s_1, \dots, s_l) = (c_k^{b_1}(sk_1, r), \dots, c_k^{b_l}(sk_l, r))$$

5.3.4 Überprüfung

Um die Signatur zu überprüfen, wird $b = (b_1, \dots, b_l)$ von M rekonstruiert.

Wenn $pk = (c_k^{w-1-b_1}(s_1), \dots, c_k^{w-1-b_l}(s_l))$, dann ist die Signatur gültig.

W-OTS+ stellt ein Sicherheitsniveau von mindestens $n-w-1-2\log(lw)$ bits bereit[3]. Eine typische Signatur, bei der $w=16$ und die SHA-256 benutzt ($n=m=256$), ist ln Bits or 2.1 kB.

6 Merkle-Signaturverfahren

Während Einmalsignaturen eine zufriedenstellende kryptographische Sicherheit für die Unterzeichnung und Überprüfung von Transaktionen bieten, haben sie den großen Nachteil, dass sie nur einmal sicher verwendet werden können. Wenn eine Adresse auf einer Veränderung des öffentlichen Schlüssels eines einzelnen OTS-Schlüsselpaares basiert, würde dies zu einem extrem restriktiven Blockchain-Ledger führen, in dem sich alle Guthaben einer sendenden Adresse mit einer einzigen Transaktion bewegen müssten – andernfalls wären diese Guthaben dem Risiko eines Diebstahls ausgesetzt.

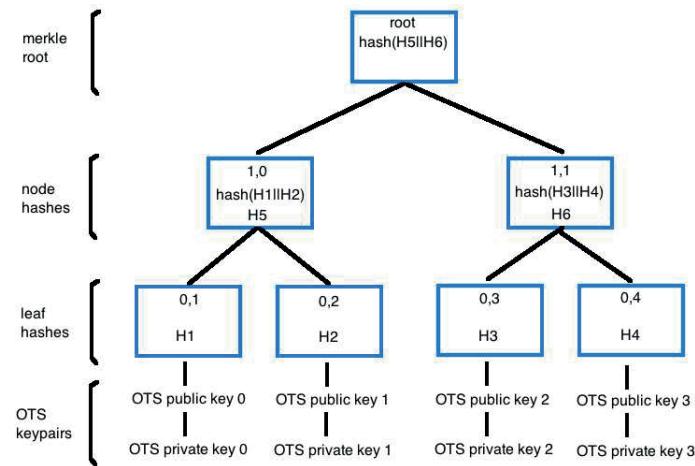
Eine Lösung besteht darin, das Signaturschemata zu erweitern, umso mehr als eine gültige OTS-Signatur für jede Ledger-Adresse generieren zu können. Dies führt dazu, dass so viele Signaturen wie OTS-Schlüsselpaare vorgeneriert werden können. Ein binärer Hash-Baum, der auch als Merkle-Baum bekannt ist, ist ein logischer Weg, dies zu erreichen.

6.1 Binary hash tree

Die Grundidee hinter einem Merkle-Baum ist ein umgekehrter Baum, der aus übergeordneten Knoten (Elternknoten) besteht, die durch das verkettete Hashing der untergeordneten Knoten (Geschwisterknoten) nach oben in verschiedenen Ebenen bis zur Wurzel berechnet werden. Die Existenz eines jeden Knotens oder Blatts kann kryptographisch bewiesen werden, indem man die Wurzel berechnet.

Ein Merkle-Baum wird aus n Grundblättern gebildet und hat eine Höhe $h(n= 2^k)$ zur Merkle-Wurzel – beginnend mit den Blätter-Hashes (Ebene 0) werden alle Ebenen der Knoten nach oben durchgerechnet. Jeder Blattknoten wird in unserem hypothetischen Ledger- Use-Case erstellt, indem ein vorher zufällig generiertes und öffentliches OTS-Schlüsselpaar gehashed wird. Anhand des Baumes unten kann man sehen, dass der Knoten oberhalb eines jeden Blatt-Hashes aus der Verkettung der ungeordneten Hashes gebildet wird.

Abbildung 3. Beispiel des Merkle-Baum Signaturschemas



Dies setzt sich nach oben fort, bis zum Zusammentreffen im Wurzel-Hash des Baumes, auch bekannt als der „Merkle-Wurzel“.

Im Beispieldiagramm können, unter Berücksichtigung der Merkle-Wurzel als öffentliches Schlüsselpaar, vier vordefinierte OTS-Schlüsselpaare verwendet werden, um vier kryptographisch sichere und valide Einmalsignaturen zu generieren. Die Merkle-Wurzel des binären Hash-Baums kann in eine Ledger-Adresse umgewandelt werden (möglicherweise durch wiederholtes Hashing mit einer angehängten Prüfsumme).

Eine vollständige Signatur S einer Nachricht M für ein gegebenes OTS-Schlüsselpaar enthält: die Signatur s , die OTS-Schlüsselnummer n , und den Merkle-Authentifizierungspfad. Bspw. gilt für das OTS-Schlüsselpaar 0 ($n = 0$):

$$S = s, n, \text{OTS public key } 0, H1, H2, H5, H6, \text{root}$$

Wenn das öffentliche OTS-Schlüsselpaar und der Blatt-Hash von s abgeleitet werden können und Elternknoten von ihren Kinderknoten errechnet werden können, kann dies faktisch verkürzt werden auf:

$$S = s, n, H2, H6, \text{root}$$

wobei S gültig ist, wenn der öffentliche OTS-Schlüssel von s und M verifiziert wurde und anschließend die Hashes vom Merkle-Authentifizierungspfad eine passende Merkle-Wurzel (öffentlichen Schlüssel) ergeben.

6.2 Status

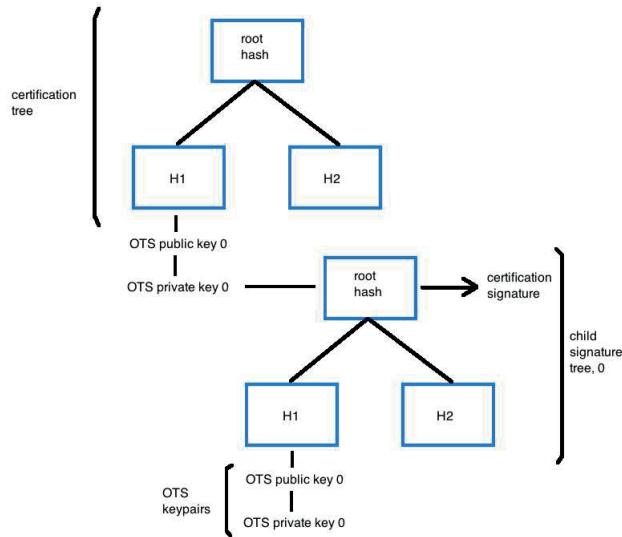
Die Verwendung des oben genannten Merkle-Signatur-Schemas (MSS) vertraut vollständig darauf, dass OTS-Schlüssel nicht wiederverwendet werden. Sie ist daher abhängig vom Zustand der Signaturen oder der signierten Transaktionen, die aufgezeichnet wurden. Dies würde im wirklichen, realen Betrieb möglicherweise ein Problem darstellen. Aber der unveränderliche, öffentliche Blockchain-Ledger ist jedoch ein ideales Speichermedium für ein zustandsorientiertes, kryptographisches Signaturschema. Ein neueres, hash-basiertes, kryptographisches Signaturschema namens SPHINCS, welches praktisch zustandslose Signaturen mit 2^{128} Bit-Sicherheit bietet, wurde 2015 veröffentlicht [2].

6.3 Hypertrees

Ein Problem der grundlegenden MSS ist, dass die Anzahl der Signaturen möglicherweise limitiert ist und alle OTS-Schlüsselpaare vorgeneriert werden müssen, bevor der Merkle-Baum berechnet werden kann. Die Zeit, die zur Schlüsselerzeugung und zur Signatur aufgewendet wird, steigt exponentiell mit der Höhe des Baums h an, was bedeutet, dass Bäume größer als 256 OTS-Schlüsselpaare mehr Zeit und Rechenkapazität bei der Erzeugung beanspruchen.

Eine Strategie, um die Rechenleistung während der Schlüssel- und Baumgeneration zu verschieben und die Anzahl der verfügbaren OTS-Schlüsselpaare zu erhöhen, ist es, einen Baum zu verwenden, der selbst aus Merkle-Bäumen besteht, ein sogenannter Hyperbaum. Die Idee dahinter ist, die Merkle-Wurzel eines Kindbaumes mit einem OTS-Schlüssel aus dem Blatt-Hash eines Eltern-Merkle-Baums zu kennzeichnen, der als Zertifizierungsbaum bekannt ist.

Figur 4. Merkle-Bäume verbinden



In der einfachsten Form ($\text{Höhe}, h=2$) wird ein Zertifizierungsbaum mit 21 OTS-Schlüsselpaaren vorberechnet. Wenn die erste Signatur benötigt wird, wird ein neuer Merkle-Baum (Signaturbaum 0) errechnet und mit einem der OTS-Schlüsselpaare des Zertifizierungsbaums signiert. Der Signaturbaum besteht aus n Blatt-Hashes mit den entsprechenden OTS-Schlüsselpaaren und dient dazu, Mitteilungen je nach Bedarf zu signieren. Wurde jedes OTS-Schlüsselpaar im Zertifizierungsbaum verwendet, wird der nächste Zertifizierungsbaum (Signaturbaum 1) vom zweiten OTS-Schlüsselpaar des Zertifizierungsbaums signiert und der nächste Signaturen-Durchlauf ist möglich.

Die Signatur S einer solchen Hyperbaum-Konstruktion wird etwas komplizierter und beinhaltet folgendes:

1. aus dem Signaturbaum: $s, n, \text{ Merkle Pfad, Wurzel}$
2. von jedem Zertifizierungsbaum: s (des Kindbaumes der Merkle-Wurzel), $n, \text{ Merkle Pfad, Wurzel}$

Es ist theoretisch möglich, Ebenen dieser Bäume vom Zertifizierungsbaum herab zu verschachteln, um die originale MSS unendlich zu erweitern. Die Größe der Signatur erhöht sich jedoch linear für jeden weiteren signierten Baum, während die Signaturkapazität des Hyperbaums exponentiell steigt.

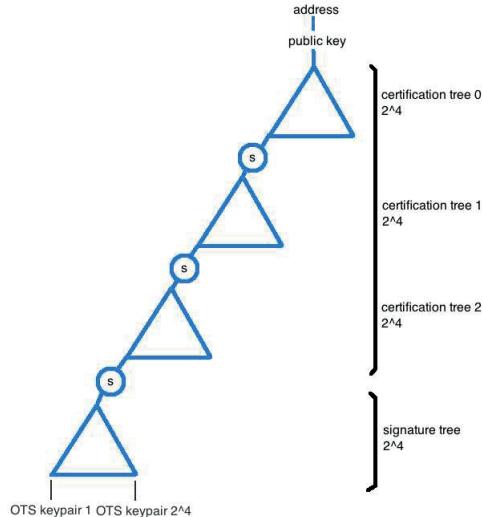
6.3.1 Beispiel eines Hypertree

Um zu demonstrieren, wie einfach das MSS durch eine Hyperbaum-Konstruktion erweitert werden kann, stellen Sie sich einen initialen Zertifizierungsbaum der Höhe $h_1 = 5$, mit 2^5 Blatt-Hashes und den dazugehörigen OTS-Schlüsselpaaren vor. Die Merkle-Wurzel dieses Baumes wird verändert, um eine Ledger-Adresse zu generieren. Ein weiterer Merkle-Baum, ein Signaturbaum in identischer Größe ($h_2 = 5$, 2^5 Blätter und OTS-Schlüsselpaare) wird instanziert. 32 Signaturen sind möglich, bevor der nächste Signaturbaum erstellt werden muss. Die Gesamtanzahl der möglichen Signaturen beträgt $2^{h_1+h_2}$, was in diesem Fall $2^{10} = 1024$ entspricht.

Auf einem MacBook Pro 2.7 GHz i5 mit 8 GB RAM erzielt die Erstellung eines OTS-Schlüsselpaars und eines Merkle-Zertifizierungsbaums in verschiedenen Größen folgende Resultate (nicht optimierter Python-Code, Winterntz OTS): $2^4 = 0.5s$, $2^5 = 1.2s$, $2^6 = 3.5s$, $2^8 = 15.5s$. Ein Hyperbaum benötigt für die initiale Generierung von zwei 2^4 -Bäumen etwa 1s im Vergleich mit den 15.5s für einen Standard 2^8 MSS-Baum mit derselben Signaturkapazität.

Das Erhöhen der Tiefe (oder Höhe) eines Hyperbaums verstärkt diesen Trend. Ein Hyperbaum, der aus vier verketteten 2^4 Zertifizierungsbäumen besteht und einem Signaturbaum mit der Größe 2^4 , ist in der Lage, $2^{20} = 1,048,576$ Signaturen zu erstellen. Dies geht mit einem erhöhten Aufwand an Signaturgröße, jedoch einer vermindernden Erstellungszeit von 2.5 s einher.

Abbildung 5. Aufbau eines Hyperbaums



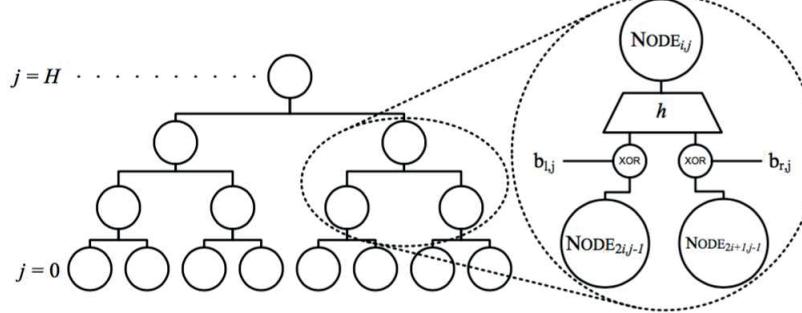
Es ist nicht erforderlich, dass ein Hyperbaum symmetrisch verläuft, sodass dieser, wenn er anfangs mit zwei Bäumen erstellt wurde, später jederzeit durch das Signieren weiterer Ebenen erweitert werden kann. Signaturen einer Ledger-Adresse würden daher niedrig beginnen und wachsen, wenn sich die Tiefe des Baums erhöht. Die Verwendung eines Hyperbaums zur Erstellung und Signierung von Transaktionen einer Ledger-Adresse benötigt vermutlich nie $>2^{12}$ Transaktionen. Daher ist die Möglichkeit, mit rechnerischer Leichtigkeit 2^{20} Mal sicher bei einer Hyperbaum-Tiefe von $h = 5$ zu signieren, mehr als ausreichend.

6.4 XMSS

Buchmann et al. veröffentlichten 2011 erstmalig zum erweiterten Merkle-Signatur-Schema (XMSS).

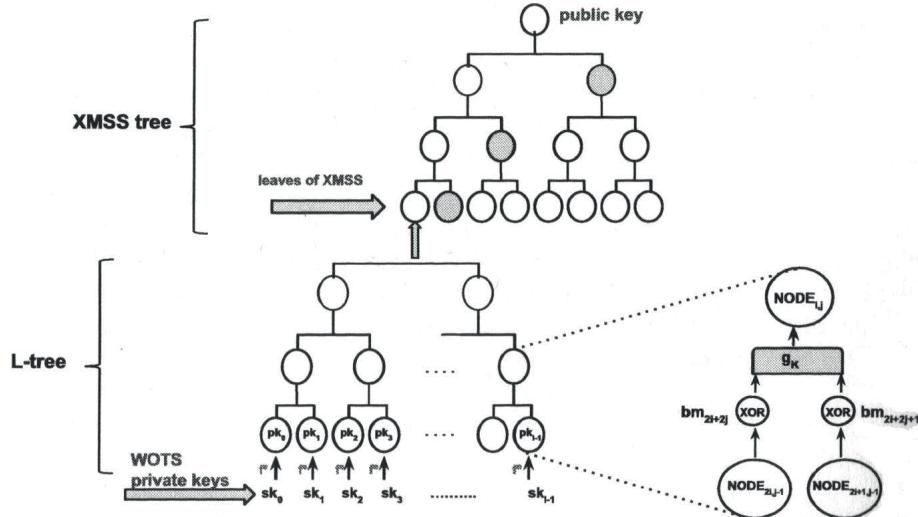
Anschließend wurde das Verfahren in einem IETF-Entwurf letztes Jahr beschrieben[4][7]. Es ist nachweislich zukunftssicher und unter den gewählten Nachrichtenattacken mit minimalen Sicherheitsanforderungen existentiell unveränderbar: eine gegenüber PRF- und Second-Pre-Image-Attacken resistente Hash-Funktion. Das Schema ermöglicht die Erweiterung von Einmalsignaturen über einen Merkle-Baum, aber mit dem elementaren Unterschied, dass die XOR-Bitmaske der untergeordneten Knoten (child nodes) verwendet werden kann, bevor die Hashes im übergeordneten Knoten (parent nodes) miteinander verkettet werden. Die Verwendung der XOR-Bitmaske erlaubt das Ersetzen der kollisionsresistenten Hashfamilie.

Fig. 1. The XMSS tree construction



Die Blätter des Baumes sind ebenfalls keine OTS-Schlüsselpaar-Hashes, sondern stellen die Wurzel des untergeordneten L-Baums dar, welche die öffentlichen OTS-Schlüssel enthält, 1 Stücke bilden die Basis-Blätter. Für die Einmalsignaturen wurde Winternitz OTS+ verwendet (obwohl die 2011-Variante zuerst beschrieben wurde).

Abbildung 7. XMSS-Konstruktion [8]



Die Bitlänge des öffentlichen XMSS-Schlüssels beträgt $(2(H + \lceil \log l \rceil) + 1)n$ eine XMSS-Signatur hat die Länge $(l+H)n$ und die Länge des privaten Signaturschlüssels ist $< 2n$.

Buchmann beschreibt die Performance mit einem Intel® i5 2.5GHz für einen XMSS Baum mit der Höhe $h= 20$ und $w= 16$ und der verwendeten, SHA-256 Hash-Funktion ($m = 256$) bei ca. einer Millionen Signaturen. Mit identischen Parametern und Hardware dauert die Signierung 7 ms, die Verifikation 0.52 ms und die Generierung des Schlüssels 466 Sekunden. Das Sicherheits-Level, das mit diesen Parametern erreicht wurde, beträgt 196 Bits bei einer Größe des öffentlichen Schlüssels von 1.7 kb, einem privaten

von 280 Bits und einer Signatur von 1.8 kb. XMSS ist ein attraktives Schema, hat aber den Nachteil, dass sehr viel Zeit benötigt wird, um einen Schlüssel zu generieren.

6.4 XMSS Baumperformance

Die Bildung eines 4096 Blatt XMSS-Baumes ($h = 12$) mit allen Schlüsseln und Bitmasken, die aus einer hash-basierten PRF generiert wurden, dauerte, auf der selben Hardware wie oben beschrieben (MacBook Pro 2.7GHz i5, 8GB RAM) und unter Verwendung einer nicht-optimierten Python-Bibliothek, welche für einen QRL Testnode konzipiert wurde, 32 s. Inkludiert ist die Generierung von mehr als 8000 Bitmasken und über 300.000 sk-Fragmenten via PRF. Zur dramatischen Leistungsverbesserung tragen sowohl ein effizienter Merkle-Baum-Traversal-Algorithmus als auch die Notwendigkeit bei, nur $w-1$ Hashes je Kettenfunktion in Bezug auf private Schlüssel in WOTS+ entgegen einer 2^{w-1} mit herkömmlichen WOTS durchführen zu müssen.

Eine vollständige Signaturgröße von etwa 5.75 kb wurde in dieser Konstruktion (11.75 kb hexadezimale String-Encodierung) erreicht, einschließlich des OTS-Schlüsselpaares n , der Signatur, der XMSS-Authentifizierungsroute, dem öffentlichen OTS-Schlüssel und dem öffentlichen Schlüssel des XMSS-Baumes (inkl. des Wertes für den öffentlichen PRF-Schlüssel und den XMSS-Baum-Root).

Für Bäume mit unterschiedlichen Signaturkapazitäten, welche durch PRF mit einem zufälligen Wert berechnet wurden, wurde die folgende Performance erreicht: ($h=9$) 512 4.2s, ($h=10$) 1024 8.2s, ($h=11$) 2048 16.02s.

7 Empfohlenes Signaturschema

7.1 Sicherheitsmaßnahmen

Bei der Entwicklung des QRL ist es wichtig, dass die kryptographische Sicherheit des Signaturschemas gegenüber klassischen- und Quantencomputerangriffen sicher ist, sowohl in der heutigen Zeit als auch in den kommenden Jahrzehnten. XMSS bietet, unter Verwendung von SHA-356, wobei $w = 16$ ist, 196-bit Sicherheit mit prognostizierter Sicherheit gegenüber Brute-Force-Attacken bis etwa in das Jahr 2164[9].

7.2 QRL Signaturen

Die Verwendung eines erweiterbaren, zukunftsorientierten und asymmetrischen Hypertree-Signaturschemas, bestehend aus verketteten XMSS-Bäumen, wird empfohlen. Dies hat den doppelten Vorteil der Verwendung eines validierten Signaturschemas und ermöglicht die Erzeugung von Leger-Adressen mit der Fähigkeit, Transaktionen unter Vermeidung von langen Vorberechnungsverzögerungen zu signieren, welche in größeren XMSS-Konstruktionen bereits beobachtet werden konnten. W-OTS+ entspricht daher dem ausgewählten hash-basierten Einmal-Signaturschema sowohl aus Performance- als auch aus Sicherheitsgründen.

7.3 Konstruktion des Hypertree

7.3.1 Schlüssel und Signaturgrößen

Da die Anzahl der Bäume innerhalb eines Hypertree stetig wächst, wachsen auch die Schlüssel- und Signaturgrößen linear an – wobei jedoch die Signaturkapazität exponentiell ansteigt. Die Größen für verschiedene XMSS-Bäume, abgeleitet von den öffentlichen Schlüsseln und Signaturen (basierend auf der Beschreibung aus 2011), wobei $w = 16$, $m = 256$, h die Höhe des Baumes und SHA-256 als ausgewählter, kryptographischer Hash-Algorithmus verwendet wurde, sind:

- $h = 2$, 2^2 Signaturen: Öffentlicher Schlüssel 0.59kb, signature 212kb (0.4s)
- $h = 5$, 2^5 Signaturen: Öffentlicher Schlüssel 0.78kb, signature 2.21kb (0.6s)

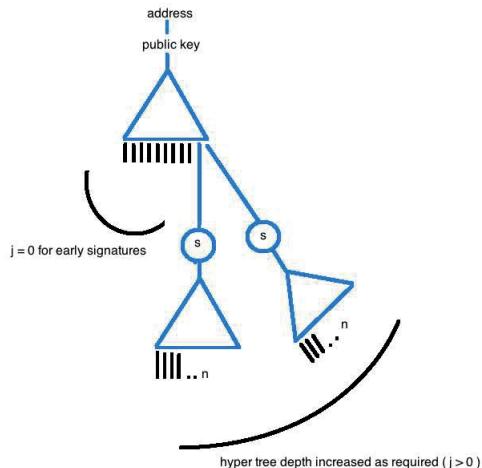
- $h = 12$, 2^{12} Signaturen: Öffentlicher Schlüssel 1.23kb, Signatur 2.43kb (32s)
- $h = 20$, 2^{20} Signaturen: Öffentlicher Schlüssel 1.7kb, Signature 2.69kb (466s[3])

Der Konflikt zur Erstellung eines XMSS-Hypertree (4 Bäume, $j = 3$, $h = 5$) mit einer eventuellen Signaturkapazität von 2^{20} in weniger als 3s, verglichen mit 466s für eine Signatur mit einer Größe von 8.84 kb gegenüber 2.69 kb, ist ggf. akzeptabel.

7.3.2 Asymmetrie

Durch die Erstellung eines asymmetrischen Baumes können frühe Signaturen mit einer einzigen XMSS-Baumkonstruktion erfolgen, welche später nach Bedarf und Aufwand für weitere Signaturen erweitert werden kann. Das Ziel dabei ist, dass dies wahrscheinlich ohne Konsequenzen für die Blockchain-Ledger-Anwendung durchgeführt werden kann und das Wallet einem Benutzer die Option zwischen Signaturkapazität versus Signaturgröße und Schlüssellänge geben kann. Eine maximale Baumtiefe von $j = 2$ sollte unter allen Umständen ausreichen.

Abbildung 8. Asymmetrischer Hypertree



7.4 Spezifikationen des QRL-Hypertree

Die folgenden Standardparameter sind für eine Standard-Hypertree-Konstruktion anzuwenden:

- $j = 0$ ($j \in \{0 \leq x \leq 2\}$), $h = 12$ ($h \in \{1 \leq x \leq 14\}$), oberes Band der möglichen Signaturen: 2^{36} minimale Signaturgröße: 2.21kb, maximale Signaturgröße: 7.65kb.

Beispiel: ein einzelner XMSS-Baum, $h = 12$ mit 4096 verfügbaren Signaturen, welcher mit weiteren Bäumen auf bis zu $h = 14$ erweitert werden könnte. Bei den meisten Benutzern ist es unwahrscheinlich, dass überhaupt zusätzliche Bäume benötigt werden.

7.4.1 Beispiel QRL-Signaturen

Bei einer vorliegenden komplexen Hypertree-Konstruktion, bei der $j = 2$ und $h = 14$ ist, benötigt die Signatur einer Nachricht m , wobei n die Position des OTS-Schlüsselpaares für jeden XMSS-Baum ist:

- Signaturbaum, $j = 2$: OTS Signatur von m, n , Merkle Authentifizierungsnachweis, Merkle-Wurzel des Signaturbaums
- Zertifizierungsbaum $j = 1$: OTS Signatur der Merkle-Wurzel des Signaturbaums ($j = 2$), n , Merkle-Authentifizierungsnachweis, Merkle-Wurzel
- Originaler XMSS Baum, $j = 0$: OTS Signatur der Merkle-Wurzel ($j = 1$), n , Merkle-Authentifizierungspfad, Merkle-Wurzel

Zum Verifizierungsvorgang gehört die Generierung des öffentlichen OTS-Schlüssels aus m und der Signatur sowie der anschließenden Bestätigung, dass der Merkle-Authentifizierungsnachweis die Merkle-Wurzel des Signaturbaums ergibt. Dies wird die Nachricht der nächsten OTS-Signatur. Hieraus wird der nächste öffentliche OTS-Schlüssel generiert – der mitgelieferte Authentifizierungsnachweis wird verwendet, um die Merkle-Wurzel des Zertifizierungsbaums zu erstellen, welcher dann wieder zur Nachricht der nächsten OTS-Signatur des Zertifizierungsbaums wird usw. Eine Signatur ist nur dann gültig, wenn die Merkle-Wurzel des höchsten Baums, des originalen XMSS-Baums, ($j = 0$) korrekt generiert worden ist.

Bemerkenswert ist, dass die öffentlichen OTS-Schlüssel nicht zur Verifikation der XMSS-Baumsignatur erforderlich sind. Tatsächlich kann die Merkle-Wurzel für jeden Baum errechnet werden und daher bei der Verifizierung der Hypertree-Signatur übersprungen werden, wenn die sendende Ledger-Adresse bekannt ist (da sie ein errechnetes Derivat der Merkle-Wurzel für den höchsten XMSS Zertifizierungsbaums ($j = 0$) innerhalb der QRL-Signatur ist – siehe „Accounts“)

Da das Signaturschema zukunftsorientiert ist, muss die Wallet-Implementierung n für jeden XMSS-Baum, der im Hypertree für eine bestimmte Adresse generiert wird, beibehalten und aktualisiert werden.

7.5 PRF

PRF vom Seed. HMAC_DRBG.

7.6 Deterministisches Wallet

Durch die Verwendung eines einzelnen SEED ist es möglich, einen sehr großen XMSS-Baum zu generieren, welcher für die meisten Benutzer auf lange Sicht ausreichend sein sollte. Eine sichere Entropie-Quelle wird verwendet, um diesen SEED zu generieren, welcher von einer sicheren PRF-Funktion zurückgegeben wird, um ein Set von pseudozufälligen Schlüsseln zu generieren, welche wiederum den Baum generieren. Ein Nachteil der Verwendung desselben XMSS-Baums ist, dass der Benutzer darin beschränkt wird, nur eine Adresse zu nutzen (obwohl Offenlegung öffentlicher Schlüssel kein Problem bei einem MSS darstellt).

Eine Bitcoin- oder Ethereum-Adresse wird vom assoziierten öffentlichen Schlüssel abgeleitet und kann daher, als einzelner privater oder öffentlicher Schlüssel, nur eine einzige Adresse erstellen. Eine XMSS-Adresse wird vom öffentlichen Schlüssel PK abgeleitet, welcher die Merkle-Wurzel und den öffentlichen SEED enthält.

Wenn der SEED konstant bleibt, sich aber die Anzahl der OTS-Schlüsselpaare, um den Baum zu errechnen, ändert, wird sich auch die Merkle-Wurzel für jede Variation verändern. Daher wird sich nach jeder einzelnen Addition oder Subtraktion eines einzelnen OTS-Schlüsselpaares die abgeleitete Adresse ändern.

Dieses Feature sollte ggf. nur von der Wallet-/Node-Software verwendet werden, um zahlreiche Variationen des XMSS-Baums zu erstellen (Erweitern/Verkürzen des Baumes je nach Bedarf unter Verwendung desselben initialen SEEDES). Auf diese Art und Weise können so viele eindeutige Adressen generiert werden wie benötigt werden. Diese Informationen auf eine sichere, zukunftsorientierte und kompakte Weise zu sichern, ist, vom rechnerischen Standpunkt gesehen, trivial.

8 Designparameter einer Kryptowährung

Der Rest dieses Whitepapers zeigt die vorgeschlagenen Gestaltungsparameter für den QRL-Ledger auf. Der Fokus des Ledgers liegt auf einer öffentlichen Blockchain, welche höchst sicher gegenüber klassischen- und Quantencomputerangriffsvektoren ist. Da dies ein erster Entwurf ist, können sich die gegebenen Aspekte zu jeder Zeit ändern.

8.1 Proof-of-Stake

QRL soll ein quelloffener, öffentlicher Blockchain-Ledger werden, der durch einen Proof-of-Stake-Algorithmus abgesichert wird. Eine Epoche hält normalerweise für 10.000 Blocks an. Stakes werden durch die Stake-Transaktionen der vorhergehenden Epoche bestimmt. Die Grundidee ist, dass jede Validierung eines Stakes eine Transaktion mit dem letzten Hash einer iterativen Kette in der Länge von 10.000 Hashes signiert (eine XOR-Bitmaske kann während jeder Iteration angewendet werden, um die Sicherheitsanforderungen der Hash-Funktion zu minimieren). Dadurch, dass die Stake-Transaktionen in jeder Kette bestätigt werden, kann nun jeder Node im Netzwerk die kryptographische Identität der Stake-Adresse mit der Hash-Kette für die nächste Epoche verbinden.

8.1.1 Gestaltung und Zufälligkeit

Für jeden Block offenbart jeder validierende Node, der die aktuelle Epoche verarbeitet, den nächstfolgenden vorhergehenden Hash in der Kette, um die kryptographische Beteiligung nachzuweisen und für den gewinnenden Blockselektor abzustimmen.

HMAC-DRBG wird verwendet, um eine pseudozufällige Nummernsequenz von 32 Byte aus den Seed-Daten der Blockchain zu generieren (initial der Genesis-Block, später dann die hinzugefügte Entropie aus den verketteten, je vorhergehenden Block-Header-Hashes für jede nachfolgende Epoche).

Daher wird der Block, den der Stake-Validator als Block-Selektor bestimmt, ausgewählt, indem bestimmt wird, welcher Hash in seiner Nummer dem PRF-Output dieses Blocks am nächsten liegt.

Dies ist verhältnismäßig schwierig durchzuführen, da die PRF den Stakers bei Beginn der Generierung der Hash-Ketten unbekannt ist. Weiterhin ist eine iterative (verschlüsselte) Hash-Kette im Grunde genommen eine zufällige Nummernsequenz. Schließlich können die Stake-Validators selbst dann, wenn sie in irgendeiner Art zusammenarbeiten, die Inhalte der Hash-Ketten der anderen Stake-Validatoren nicht kennen, da diese noch nicht offengelegt sind.

Um zu verhindern, dass eine Block-Withholding-Attack begangen wird, wird der Fehler, einen gültigen Block zu produzieren, nachdem die Übermittlung eines gültigen Reveal-Hashs durchgeführt wurde, mit dem Verlust der gesamten Block-Belohnung für diese Adresse assoziiert. Diese wird anschließend für einen gewissen Zeitraum an der Teilnahme gehindert.

Um eine Sybil-Nodes-Attacke mit einem leeren Block oder Stake-Validator-Adressen mit einem geringen Guthaben abzuwehren, wird eine flexible Stake-Validator-Schwellenwert-Liste implementiert. Die Blockbelohnung wird in gewichteter Form ausgeschüttet, abhängig vom Guthaben der Stake-Adressen. Mit der offengelegten Hash-Nachricht gibt jeder Node auch einen Merkle-Baum-Root-Hash einer sortierten Liste von tx-Hashes in ihrem Transaktionspool weiter, zusammen mit der Anzahl der Transaktionen, die auf einen Block warten. Jeder Node schneidet eine Prozentanzahl vom oberen und vom unteren Ende ab, um zu sehen, wie viele Transaktionen im nächsten Block erwartet werden. Ist der Block leer oder enthält weniger Einträge als erwartet, erlaubt es die Anzahl der Stake-Validatoren, Aufträge in jedem Block nach oben zu verschieben (mit Ausnahme der Stake-Validatoren von arm bis reich). Wenn sich die Blockselektor-Nodes „ehrlich“ verhalten, ist das Gegenteil wahr und die Anzahl der erlaubten, teilnehmenden Stake-Validatoren steigt. Guthaben dürfen sich, in der Epoche, in der sie verarbeitet werden, nicht bewegen - dies unterbindet Versuche, die Blockauswahl durch die Erstellung von unzähligen Sybil-Stake-Validator-Adressen zu manipulieren.

8.2 Gebühren

Die größeren Transaktionsformate setzen, im Vergleich zu anderen Ledgern voraus, dass bei jeder Transaktion eine Transaktionsgebühr gezahlt werden muss. Der Autor ist der Meinung, dass künstliche Gebührenerhebungen unnötig sind und damit dem Ideal eines quelloffenen, öffentlichen Blockchain-ledgers entgegenstehen. Jede Transaktion, für die eine Mindestgebühr notwendig ist, sollte so gültig sein wie jede andere auch. Die Minimalgebühr, die Miner akzeptieren, sollte variabel sein und sich am Markt orientieren. Beispielsweise sollten Nodes und Miner die wettbewerbsfähige untere Grenze der Gebühren untereinander selber festlegen. Ein absolutes Minimum wird auf Protokollebene durchgesetzt. Daher werden Miner die Transaktionen vom Mempool nach eigenem Bedarf zur Inklusion anweisen.

8.3 Blöcke

8.3.1 Blockzeiten

Die Verarbeitungszeit bei Bitcoin zwischen zwei Blöcken beträgt etwa 10 Minuten. Dies kann, je nach Anlass, zu einer relativ langen Zeitdauer führen, bevor der nächste Block gemined wird. Neuere Ledger-Designs wie Ethereum haben dies verbessert und profitieren von einer sehr viel kürzeren Blockzeit (15 Sekunden), ohne einen Verlust von Sicherheit oder einer Zentralisierung von Minern in Bezug auf verwaisten oder verbrauchten Blöcke. Ethereum verwendet eine modifizierte Version des Haeviest

Haeviest Observed Subtree-Protokolls, welches es verwaisten oder verbrauchten Blöcken erlaubt, in der Blockchain inkludiert und honoriert zu werden[13, 5].

Da für das QRL geplant ist, direkt von Beginn an einen Proof-of-Stake-Algorithmus zu verwenden rechnen wir damit, sicher eine Blockzeit von 15 bis 30 Sekunden sicher verwenden zu können.

8.3.2 Blockbelohnungen

Jeder neu erstellte Block wird eine erste ‘coinbase’ Transaktion beinhalten, welche eine Mining-Adresse enthält, an welche eine Belohnung gleich der Summe der Coinbase-Belohnung und der kombinierten Summe der Transaktionsgebühren innerhalb des Blocks gewährt wird.

Die Belohnung für einen Block wird beim Mining eines jeden Blocks neu errechnet und folgt dem Coin-Emissionsplan.

8.3.3 Blockgrößen

Um eine Kontroverse zu vermeiden, wurde eine auf dem Bitpay-Modell basierende Out-of-the-Box-Lösung verwendet, um die Größe der Blöcke anhand eines vielfachen von x , der durchschnittlichen Größe y und der Größe des letzten Blocks z zu bestimmen[12]. Die Verwendung des Durchschnitts verhindert es, dass Miner entweder leere oder übervolle Blöcke verwenden, um die durchschnittliche Blockgröße zu verändern. x und z wären dann die Konsensus-Regeln, die das Netzwerk beachtet.

Eine maximale Blockgröße b könnte dann wie folgt einfach berechnet werden:

$$b = xy$$

8.4 Währungseinheit und Bezeichnung

Das QRL wird einen monetären Token, das *quantum* (Plural *quanta*), als Basiswährungseinheit verwenden. Jedes *quantum* ist in kleinere Einheiten wie folgt teilbar:

- 1 : Shor
- 10^3 : Nakamoto
- 10^6 : Buterin
- 10^{10} : Merkle
- 10^{13} : Lamport
- 10^{16} : Quantum

Daher ist jede Transaktion, die ein Bruchteil eines *quantum* enthält, eigentlich eine sehr große Anzahl von *Shor*-Einheiten. Transaktionsgebühren werden in *Shor*-Einheiten berechnet und gezahlt.

8.5 Konten

Benutzerguthaben werden in Konten aufbewahrt. Jedes Konto ist eine simple und eindeutige wieder-verwendbare Ledger-Adresse, bestehend aus einem String und mit dem Buchstaben ’Q’ beginnend.

Eine Adresse wird erstellt, indem ein SHA-256 der Merkle-Wurzel des höchsten XMSS-Zertifizierungsbaums erstellt wird. Eine vier Byte große Prüfsumme wird diesem Wert angehängt (erstellt aus den ersten vier Byte eines doppelten SHA-256 Hashs der Merkle-Wurzel) und es wird der Buchstabe ’Q’ vorangestellt. In Pseudocode lautet diese Adresse:

$$Q + sha256(merkle_root) + sha256(sha256(merkle_root))[: 4]$$

Eine gewöhnliche Kontoadresse wäre:

Qcea29b1402248d53469e352de662923986f3a94cf0f51522bedd08fb5e64948af479

Jedes Konto weist ein Guthaben auf, dass in *quanta* angegeben wird und in einzelne Shor-Einheiten geteilt werden kann.

Dadurch, dass bei jeder Transaktion ein neues OTS-Schlüsselpaar verwendet wird und der öffentliche Schlüssel, der für jedes Konto verwendet wird, gespeichert wird, werden Adressen zustandsabhängig verwendet (diese Abhängigkeit kann reduziert werden, da sie jederzeit wieder von der Transaktionssignatur und den Nachrichten on-the-fly errechnet werden kann; dieser Vorgang wäre jedoch rechenintensiv). Ein Transaktionszähler, der als ‘nonce’ bezeichnet wird, erhöht sich bei jeder Transaktion, die von einem Konto gesendet wurde. Dies erlaubt es, Wallets, die nicht die gesamte Blockchain speichern, ihre Position innerhalb des zukunftsorientierten Merkle-Hypertree-Signaturschemas zu verfolgen.

8.6 Ausgabe der Coins

8.6.1 Historische Überlegungen

Bitcoin war die erste dezentralisierte Kryptowährung dar und stellte zu Beginn ein Experiment ohne jeden monetären Wert dar. Es war daher angebracht, die Währung vollständig durch Mining zu vertreiben. Erst kürzlich hat sich Zcash zum selben Vorgehen entschieden, einen festgelegten Prozentsatz der Coinbase Mining Rewards an ein Open Source Projekt weiterzugeben – dies führte zu starken und unvorhergesehenen Kurschwankungen.

Andere Ledger wie Etherum haben stattdessen einen großen Prozentsatz der finalen Menge an Coins als Teil eines anfänglichen Coinangebots (initial coin offering, ICO) verkauft. Dies hat den Vorteil, dass Erstanwender durch das Unterstützen des Projekts noch Gewinne erwirtschaften können, das Projekt aber zusätzlich auch selbst in der Lage ist, Mittel zur Fortsetzung und zum Wachstum zu generieren. Der ICO-Ansatz erlaubt es dem Markt auch, einfacher zu wachsen, da den Investoren ein größerer Fluss von Coins zum Kaufen und Verkaufen aus dem Genesis-Block zur Verfügung steht.

Auroracoin verfolgte im Jahr 2014 einen anderen Ansatz, indem es jedem Einwohner in Island einen gleich großen Teil des ICO anbot, während die Entwickler 50% des gesamten Coin-Angebots für sich behielten.

Andere Kryptowährungen haben entweder Bitcoin komplett kopiert oder sind mit einer anderen Codebasis vollständig neu gestartet.

8.6.2 Guthabentransfer zwischen verschiedenen Ketten

Es ist möglich, das QRL auf der Grundlage eines aktuellen Snapshot des Bitcoin-Ledgers auszugeben, welches in den QRL-Genesis-Block eingefügt wird. Die Grundidee wäre, den Benutzern eine Möglichkeit zu geben, eine einzelne ‘Import’-Transaktion durchzuführen, die eine eindeutige Nachricht und Signatur enthalten (d.h. eine zufällig erzeugte QRL-Walletadresse, die mit dem privaten Bitcoin-Schlüssel einer Adresse, die zum Zeitpunkt des Snapshot ein Guthaben aufweist, signiert wurde). Dieses Feature könnte bis zu dem Zeitpunkt aktiv bleiben, bis eine gewisse Blockhöhe erreicht wurde - anschließend würden die verbleibenden Coins normal gemined. Der initial eingefügte Snapshot im Genesis-Block würde dann bei Erreichen dieser Blockhöhe entfernt. Ein Nachteil daran ist, dass Nutzer anderer Währungen bestraft werden und es eventuell eine Herausforderung für neue Nutzer darstellt. Ein technisches Problem könnte die Tatsache darstellen, dass es möglich ist, einen öffentlichen ECDSA-Schlüssel nur aus der Signatur und einer Nachricht wiederherzustellen. Dies würde die öffentlichen Schlüssel der Bitcoin-Adressen, die diesen Prozess durchlaufen, permanent offenlegen. Um dies zu verhindern müssten die Guthaben zu einer neuen, zufällig generierten Bitcoin-Adresse transferiert werden, nachdem der Prozess abgeschlossen wurde.

(? Wäre möglich, dasselbe Feature für Etherum-Nutzer zu ermöglichen)

8.6.3 Vorgeschlagene Ausgabe – Entwurf

Die initiale Ausgabe des QRL wäre wie folgt:

- ICO (Initial Coin Offering) von 1 Millionen *quanta* (4,7% der Gesamtmenge) vor dem Start
- Ein Snapshot aller Bitcoin-Adressen, die ein Guthaben von mehr als 0.01 btc aufweisen, um den initialen QRL-Genesis-Block zu formen. Jeder, der sein Guthaben direkt 1:1 vom Bitcoin-Ledger zu QRL transferieren möchte, wäre über das Node-Wallet dazu in der Lage, bis zum Zeitpunkt, an dem die Blockhöhe 518400 erreicht wurde (3-6 Monate).

- Eine weitere Millionen *quanta* verbleiben im Genesis-Block zur Verwendung durch die Stiftung.
- Das verbleibende Angebot würde gemined ((21. 000. 000 - (2. 000. 000 + btc-Guthaben, die bis zu einer Blockhöhe von 518400 importiert wurden)

8.7 Coin Emissionsplan

Ein grundlegendes Merkmal von Bitcoin ist das Fehlen bzw. die feste Obergrenze für die Ausgabe der zugrundeliegenden Geldmittel. QRL folgt Bitcoin in dieser Hinsicht mit einer festgelegten oberen Grenze für die Ausgabe von Coins von 21×10^6 *quanta*. Bevorzugt wird ein gerader, exponentieller Verfall der Block-Belohnungen bis zur festen Obergrenze des Coinangebots. Dadurch wird die mit dem Phänomen der Bitcoin-‘Halbierung’ verbundene Volatilität eliminiert.

Das gesamte Coinangebot, $x = 21 \times 10^6$ abzüglich der Coins, die im Genesis-Block generiert wurden, y , wird sich für immer exponentiell von Z_0 verringern. Die Zerfallskurve wird berechnet, um die Entlohnung für das Minen auf etwa 200 Jahre (bis 2217 AD, 4204800000 Blöcke bei 15s Blockzeiten) zu verteilen, bis nur noch ein einziges *quanta* ungemined bleibt (auch, wenn das Minen danach weiterhin fortgesetzt werden könnte)

Das verbleibende Coinangebot des Block t , Z_t kann wie folgt berechnet werden:

$$Z_t = Z_0 e^{-\lambda t}$$

Der Koeffizient, λ wird aus $\lambda = \frac{\ln Z_0}{t}$ berechnet, wobei t die Gesamtanzahl der Blöcke im Emissionsplan bis zum finalen *quanta* ist und zusätzlich gilt: bis Block 518400, $\lambda = 3.98590885111 \times 10^{-08}$. Die Block-Belohnung b wird für jeden Block wie folgt berechnet:

$$b = Z_{t-1} - Z_t$$

Zwischen dem Genesis-Block und dem Block Nr. 518400 können Bitcoin-Guthaben via Import-Transaktionen in den Ledger transferiert werden. Ab Block 518401 wird der Emissionsplan in den neu importierten Guthaben die Sperrung rückgängig machen, Z_t reduzieren und die Block-Belohnungen entsprechend anpassen.

References

- [1] <http://oxt.me/charts>.
- [2] D. Bernstein. Sphincs: practical stateless hash-based signatures. 2015.
- [3] J Buchmann. On the security of the winternitz one-time signature scheme.
- [4] J. Buchmann. Xmss – a practical forward secure signature scheme based on minimal security assumptions. 2011.
- [5] V Buterin. Ethereum whitepaper. 2013.
- [6] A. Hulsing. W-ots+ - shorter signatures for hash-based signature schemes. 2013.
- [7] A. Hulsing. Xmss: Extended hash-based signatures. 2015.
- [8] A Karina. An efficient software implementation of the hash-based signature scheme mss and its variants. 2015.
- [9] A. Lenstra. Selecting cryptographic key sizes. 2001.
- [10] R. Merkle. A certified digital signature. *CRYPTO*, 435, 1989.

- [11] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
- [12] S Pair. A simple, adaptive blocksize limit. 2016.
- [13] Yonatan Sompolinsky. Accelerating bitcoin's transaction processing fast money grows on trees, not chains. 2014.
- [14] A. Toshi. The birthday paradox. 2013.