# Quantum Resistant Ledger (QRL)

peterwaterland@gmail.com

2016 年 11 月

Translated by ”converghub”

概要

個人の電子マネーが長寿であるためには，コンピューティングの発展に対して安全でなければならない。古典コンピューティングおよび量子コンピューティングによる攻撃に耐性のある，ハッシュベースデジタル署名を利用した暗号通貨台帳の設計と発行について発表する。

## 1 Introduction

ブロックチェーンとして記録され，Proof of Work により安全が担保されている，価値ある P2P インターネット台帳の概念は，2008 年に初めて報告された [11]。ビットコインは，今日でも最も広く利用されている暗号通貨である。それに続いて，何百もの類似する暗号通貨台帳が生み出されている。しかしながら，それらは少しの例外を除いて，トランザクションの確実な検証のため，同一の楕円曲線 DSA 署名方式（ECDSA）に依存している。今日最も一般に使用される ECDSA，DSA や RSA などの署名方式は，理論的に量子コンピューティングによる攻撃に対して脆弱である。量子コンピューティングの非線形な進歩が突然起こる潜在的脅威に対抗するため，量子コンピューティングによる攻撃に耐性のあるブロックチェーン/分散型台帳の設計と構築について探究するのは価値あることだろう。

## 2 Bitcoin Transaction Security

現在，特定のビットコインアドレスの秘密鍵（$x \in N | x < 2^{256}$）から有効な楕円曲線（SECP256k1）署名を含むトランザクションを作成することによって，ビットコインアドレスから使う（トランザクション出力を使わない）のが可能なだけだ。真に無作為に生成された秘密鍵を，秘密にしたままか紛失した場合，当然そのアドレスから全く資金を移動できなくなる。
あるビットコインの秘密鍵が衝突する確率は，$2^{256}$ 分の 1 である。任意のビットコインアドレスの衝突確率は，誕生日のパラドックスを利用することで推定できる。衝突確率が 0.1%になるように生成する必要があるビットコインアドレスの数は $5.4 \times 10^{23}$ だ [14]。

しかしながら，トランザクションが署名されると，このとき送信アドレスの ECDSA 公開鍵が明らかになり，ブロックチェーン内に格納される。最良の方法はアドレスを再利用しないことであるが，2016 年 11 月現在，ビットコイン台帳残高全体の 49.58% が公開された公開鍵を持つアドレスに保管されている [1]。

## 3 Quantum Computing Attack Vectors

RSA，DSA や ECDSA は，大きな整数の因数分解の難しさ，離散対数問題，楕円離散対数問題に基づいてセキュアであり続けている。Shor のアルゴリズム (1994) は，大きな整数の因数分解および離散対数問題を多項式時間で解くことができる。それゆえに，ECDSA 公開鍵があれば，量子コンピュータは理論的に秘密鍵を再構成しうる。228 bit の ECDSA を解くことが可能な 1300 と 1600 qubit($2^{11}$) の量子コンピュータがあれば，より短い鍵長である ECDSA は RSA よりも脆弱であると考えられる。

公開されている量子コンピュータの発展としては，まだ $2^5$ qubit で，小さい整数 (15 や 21) の因数分解しかできない。しかしながら，2015 年 8 月に NSA は量子コンピューティングを懸念して，楕円曲線暗号を表面上廃止した。現在どれほど量子コンピューティングが発展しているか，あるいは，インターネットで一般的に使用される暗号化プロトコルをポスト量子セキュアにする，この分野の何らかのブレイクスルーが公表されるかは不明瞭だ。いくらか反体制の発端があれば，ビットコインは量子コンピュータを有する敵の最初のターゲットとなりうる。

仮に量子コンピューティングの著しい進歩が公になった場合，ノード開発者は耐量子暗号署名スキームをビットコインに実装し，全てのユーザに対して ECDSA ベースのアドレス内の残高を，量子的に安全な新しいアドレスに移すことを促すだろう。影響を及ぼされるアドレスの割合を緩和するため，プロトコルレベルで公開鍵の再利用を無効にすることが理に叶うだろう。そのように計画されたアップグレードは，結果として価格変動と併せて Satoshi Nakamoto 氏所有の 100 万ビットコインを移動することになるかもしれない。

好ましくないシナリオは，量子コンピューティングが静かに非線形な発展を遂げ，それに続き量子コンピューティングにより，公開鍵の明らかなビットコインアドレスが攻撃されることだ。そのような窃盗行為は，大きな売り圧力とシステムへの信頼の完全な喪失により，ビットコインの為替価格に破壊的な影響をもたらすだろう。価値保蔵（” デジタルゴールド”）としてのビットコインの役割は，どうあってもひどく打撃を受け極端な結果になる。これに関連して，著者は，暗号通貨台帳に耐量子暗号署名を実装して，ブラック・スワンに備え非常用の価値保蔵を創造してみることが合理的だと信じている。

## 4 Quantum-resistant signatures

重要な耐量子暗号システムはいくつかある。ハッシュベース暗号，符号ベース暗号，格子ベース暗号，多変数多項式公開鍵暗号，秘密鍵暗号などである。これら全てのスキームは十分に長い鍵サイズがあれば，古典コンピューティング及び量子コンピューティングによる攻撃の両方に耐性があると考えられている。

進歩的でセキュアなハッシュベース暗号署名スキームは，暗号学的ハッシュ関数の衝突耐性に依存する最少限度のセキュリティ要件でできている。ハッシュ関数を変更することにより，新たなハッシュベースデジタル署名スキームを作ることが可能である。ハッシュベースデジタル署名はよく研究されており，将来のポスト量子暗号の一次候補に相当する。かくして，それらを QRL のポスト量子暗号署名のクラスとして選択している。

# 5 Hash-based digital signature

耐量子ハッシュベース署名は，一方向性の暗号学的ハッシュ関数に依存しており，それらはつまり，SHA-256 や SHA-512 のように，メッセージ $m$ を受け取り，固定長 $n$ のハッシュ値 $h$ を出力する。暗号学的ハッシュ関数を使用することで，$h$ から $m$（原像計算困難性），あるいは $h_2 = hash(h)$（第 2 原像計算困難性）となる $h_2$ から $h$ を総当たりするのを，計算的に実行不可能にしている。一方で，同一の $h$ を出力する二つのメッセージ $(m_1 \neq m_2)$ を見つけることはとても困難である（衝突耐性）。

グローバーのアルゴリズムは，ハッシュの衝突を見つける，あるいは $m$ を見つける原像攻撃を実行するために使用されるかもしれない。計算量は $O(2^{n/2})$ である。それゆえに，完全な暗号学的ハッシュ関数を仮定すると，128bit のセキュリティを持続するため，少なくとも 256bit のハッシュ値の長さ $n$ が選択されなければならない。

ハッシュベースデジタル署名は，検証のための公開鍵 $pk$ とメッセージの署名のための秘密鍵 $sk$ を必要とする。ブロックチェーン台帳の一部として包含される際の適合性に関して，種々のハッシュベースのワンタイム署名 (OTS) を議論する。

## 5.1 Lamport-Diffie OTS

1979 年，ランポートは $m$ bit（一般に衝突耐性ハッシュ関数の出力）メッセージのハッシュベースワンタイム署名について説明した。鍵ペア生成は，ランダムな秘密鍵の $m$ 個のペアを生み出す，つまり，$sk_j^m \in \{0,1\}^n$。ここで，$j \in \{0,1\}$ である。すなわち，秘密鍵は，$sk = ((sk_0^1, sk_1^1), .., .., (sk_0^m, sk_1^m))$ である。関数 $f$ を $pk_j^m = f(sk_j^m)$ のように $m$ 個の公開鍵のペアを生成する一方向性の暗号学的ハッシュ関数 $\{0,1\}^n \to \{0,1\}^n$ としよう。つまり，公開鍵は，$pk = ((pk_0^1, pk_1^1), .., .., (pk_0^m, pk_1^m))$ となる。署名するには $sk_j$ を選択するメッセージハッシュのビット単位の検査が含まれており（つまり，$bit = 0$ のとき $sk_j = sk_0$，$bit = 1$ のとき $sk_j = sk_1$），秘密鍵の半分が明らかな $s = (sk_j^1, .., sk_j^m)$ となる署名を生成する。署名を確かめるため，ビット単位（$j \in \{0,1\}$）のメッセージ・ハッシュの検証は，$(pk_j = f(sk_j))^m$ をチェックする。

グローバーのアルゴリズムについて 128 bit のセキュリティが望ましいと仮定しよう。そのとき，メッセージ長は，$m = 256$ かつ $n = 256$ より $pk = sk = 16$ となる，SHA256 による固定のハッシュ値の出力で，使用される OTS それぞれの 8KB 署名である。ランポート署名は一度だけ使用されるべきであり，また，とても高速に生成されるかもしれないが，大きな鍵・署名に悩まされており，トランザクション・サイズの拡張により，一般のブロックチェーン台帳にとって非実用的なものとなっている。

## 5.2 Winternitz OTS

$n$ bit の秘密鍵・公開鍵をもつ $m$ bit のメッセージ・ダイジェスト $M$，一方向性関数 $f : \{0,1\}^n \to \{0,1\}^n$，そして Winternitz のパラメータ $w \in N | w > 1$ に対して，Winternitz ワンタイム署名の概念は，暗号学的ハッシュ関数をランダムな秘密鍵のリスト $sk \in \{0,1\}^n, sk = (sk_1, .., sk_{m/w})$ に反復して適用する。そして，公開鍵 $(pk \in N | \{0,1\}^n)$，$pk_x = f^{2^w - 1}(sk_x)$，$pk = (pk_1, .., pk_{m/w})$ で終わる，長さ $w - 1$ のハッシュ・チェ

ーンが生成される。ランポート署名のメッセージ・ダイジェストのビット単位の検査と異なり，その代わりに，$i \in N, i < 2^w - 1$ という $i$ を抽出するためメッセージは一度に $w$ bit パースされる。なお，この $i$ から署名 $s_x = f^i(sk_x), s = (s_1, .., s_{m/w})$ は生成されている。パラメータ $w$ が増加すると，計算量の増加に対して，より小さい鍵と署名のトレードオフが提供される [10]。
検証には，単に $M, s$ から $pk_x = f^{2^w-1-i}(s_x)$ を生成し，公開鍵が一致しているか確認することが必要になる。

SHA-2(SHA-256) を一方向性の暗号学的ハッシュ関数 $f$ として使用する場合，$w = 8$ として $m = 256$, $n = 256$ から，$\frac{(m/w)n}{8}$bytes = 1KB の $pk = sk = s$ となる。
$pk$ を生成するのにハッシュの反復合成 $f^i$ が必要であり，ここで $i$ は，OTS 鍵ペア生成 1 回あたり $i = \frac{m}{w}(2^w - 1) = 8160$ である。$w = 16$ のときは，鍵と署名のサイズは半減するが，$i = 1048560$ となり非実用的になる。

## 5.3 Winternitz OTS+ (W-OTS+)

Buchman は，直前の反復合成 $f_k(x)$ で生成される鍵 $k$ によってパラメータ化する一方向性の反復合成写像を，乱数 $x$ に繰り返し適用することで，オリジナルの Winternitz OTS と異なるものを紹介している。これは，擬似ランダム関数 (PRF) と安全性証明が所定のパラメータで計算されたとき，適応型選択攻撃に対しても丈夫で偽造不可能なものである [3]。単純な反復合成の代わりに，関数を通じてランダムウォークを実行することで，衝突耐性ハッシュ関数族の必要性がなくなる。Hülsing はさらに異なる W-OTS+ を紹介しており，それは，反復合成写像に XOR ビットマスクを追加することで，同じ”bit” セキュリティでより小さな署名の生成を可能にしている [6]。W-OTS(2011 variant)/W-OTS+ と W-OTS のもう一つの違いは，メッセージが一度に $w$ bit ではなく $log_2(w)$ bit パースされることで，ハッシュ関数の反復計算回数を減らし，鍵と署名のサイズを大きくしていることである。

W-OTS+ を簡単に説明する。メッセージ長 $m$ に対応するセキュリティパラメータ $n \in N$，暗号学的ハッシュ関数により決定される鍵と署名，Winternitz パラメータ $w \in N | w > 1$（一般に $\{4, 16\}$）を用いて，$l$ は計算される。$l$ は WOTS+ 鍵または署名内の $n$ bit 文字列要素である。ここで，$l = l_1 + l_2$ とすると，

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

鍵付きハッシュ関数は $f_k : \{0,1\}^n \mid k \in \{0,1\}^n$ が使用される。擬似コードでは次のようになる。

$$f_k(M) = Hash(Pad(K)||Pad(M))$$

ここで，$|x| < b$ を満たす $x$ に対して，$Pad(x) = (x||10^{b|x|1})$ である。

チェーン関数 $c_k^i(x, r)$ は，$x \in \{0,1\}^n$，反復回数カウンタ $i$，鍵 $k \in K$，ランダム化要素 $r = (r_1, .., r_j) \in \{0,1\}^{n*j} \quad (j \geq i)$ を用いて，次のように定義される。

ここで，

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

Figure 1. W-OTS+ chaining function

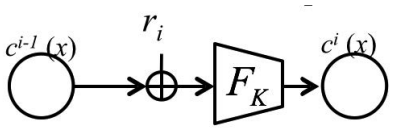
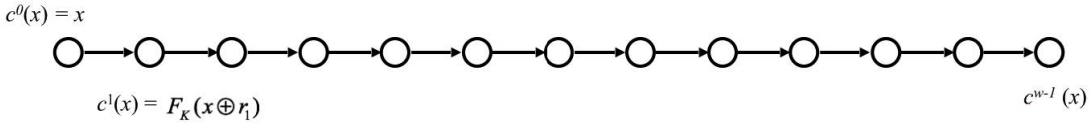



Figure 2. Example hash-chain generation




すなわち，直前の反復合成 $c_k$ とランダム化要素の排他的論理和 XOR を計算し，そのあとに $f_k$ が続き，次の反復合成 $c_k$ となる。

#### 5.3.1 Signature key

秘密鍵 $sk$ を生成するため，$(l+w-1)$ 個の $n$ ビットの文字列を一様にランダムに（擬似ランダム関数 PRF を用いて）選択し，その中の最初の $l$ 個が秘密鍵 $sk = (sk_1, .., sk_l)$ を生成し，残りの $w-1$ 個の $n$ ビットの文字列が $r = (r_1, .., r_{w-1})$ となる。関数の鍵 $k$ は一様にランダムに選択される。

#### 5.3.2 Verification key

公開鍵は，以下のようになる。

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

なお，$pk_0$ は $r$ と $k$ を含んでいる。

#### 5.3.3 Signing

署名するため，長さ $m$ のメッセージ $M$ が，$M = (M_1, .., M_{l_1})$, $M_i \in \{0, w-1\}$ となるようパースされる (M の w ベースの表現を生成)。
次に長さ $l_2$ のチェックサム $C$ が計算され，付加される。

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

$M + C = b = (b_0, .., b_l)$ とすると，署名は次のようになる。

$$s = (s_1, .., s_l) = (c_k^{b_i}(sk_1, r), .., c_k^{b_l}(sk_l, r))$$

#### 5.3.4 Verification

署名を検証するため，$b = (b_1, .., b_l)$ が $M$ により再生成される。

$pk = (c_k^{w-1-b_1}(s_1), .., c_k^{w-1-b_l}(s_l))$ のとき，署名は有効である。

W-OTS+ は少なくとも $(n - w - 1 - 2log(lw))$ bit のセキュリティレベルを提供する [3]。SHA-256($n = m = 256$) を使用する $w = 16$ の典型的な署名は，$ln$ bit あるいは 2.1KB である。

# 6 Merkle tree signature schemes

ワンタイム署名は署名とトランザクション検証に対して満足のいく暗号セキュリティを提供してくれる一方で，大きな欠点を持っており，それは一度安全に使用されるだけということだ。もし台帳アドレスが単一な OTS 鍵ペアの公開鍵の何らかの変換に基づいていた場合，これは極めて制限されたブロックチェーン台帳を導くことになる。そこでは，送信アドレスから送られた全ての資金は，一つのトランザクションごとに移動する必要がある。そうでなければ，盗まれる危険があるだろう。
解決策は，あらかじめ生成される OTS 鍵ペアと同数の署名を認める各台帳アドレスに対して，一つ以上の有効な OTS 署名を合体することで署名スキームを拡張することだ。merkle 木として知られるバイナリハッシュ木が，これを獲得するのに論理的な方法である。

## 6.1 Binary has tree

merkle 木の背景にある一般的な考え方は，連結された子ノードをルートに向かってハッシングすることで計算された，親ノードにより構成される反転木だ。どのノードまたは葉の存在も，ルートを計算することで暗号学的に証明される。

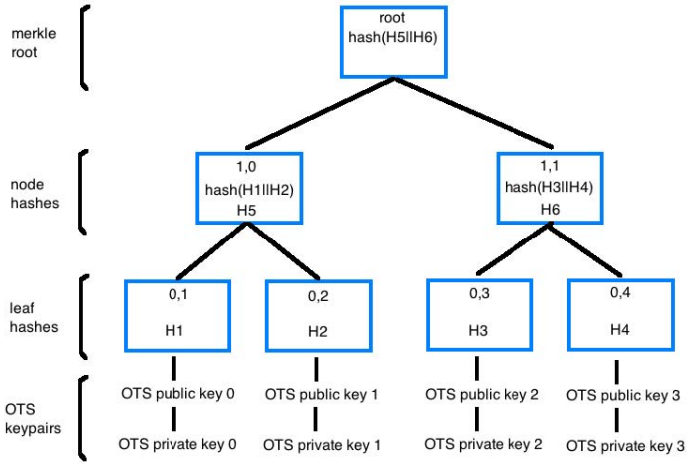
merkle 木は，ベースとなる $n$ 個の葉から成り，マークル・ルートへの高さ $h$ $(n = 2^h)$ を持つ。なお，葉のハッシュ（レイヤー 0）からノードの各レイヤーを上向きに数える。それぞれの葉のノードは，ランダムに前生成される OTS 公開鍵をハッシングすることにより，仮定した台帳ユースケース内で生成される。木の下位からは，葉のハッシュペアの上位のノードが，連結された子ハッシュをハッシングして形成されているようにみえる。

これはマークル・ルートとして知られる，木のルートハッシュに合流するまで，レイヤーを通って上に続いていく。

ダイアグラムで例示した木より，マークル・ルートを公開鍵として，４つの前計算された OTS 鍵ペアが，４つの暗号学的にセキュアで有効なワンタイム署名を生成するのに使用される。バイナリハッシュ木のマークル・ルートは，（あるいは付加されたチェックサムを用いて反復してハッシングすることで）台帳アドレスに変換される。与えられた OTS 鍵ペアに対して，メッセージ $M$ の署名全体 $S$ は，署名 $s$，OTS 鍵番号 $n$，マークル認証パスを含んでいる。つまり，OTS 鍵ペア $0(n = 0)$ に対して，

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

Figure 3. Merkle tree signature scheme example




OTS 公開鍵と葉のハッシュが $s$ から推測されるとすると，親ノードは実際それらの子ノードから計算されるので，次のように短縮される。

$$S = s, \quad n, \quad H2, \quad H6, \quad root$$

$s$，$M$ から OTS 公開鍵を検証し $S$ が有効だった場合，そのあとマークル認証パスからハッシュを確認することで，整合するマークル・ルート（公開鍵）が再生成される。

## 6.2 State

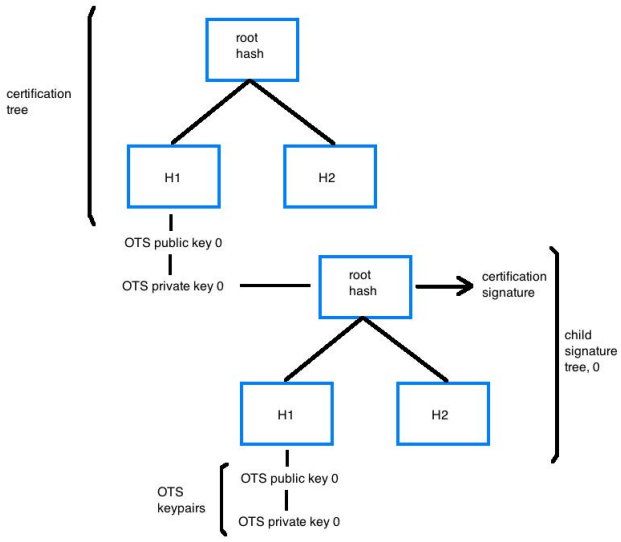
上述したマークル署名スキーム（MSS）を使用することで，確実に OTS 鍵を繰り返し使わないことになる。ゆえに，署名の状態や記録された署名済みトランザクションに依存している。通常，実社会の用途では，これは潜在的に問題となるだろう。しかし，ステートフルな暗号署名スキームにとって，変更不能なパブリックブロックチェーン台帳は理想的な記録媒体である。$2^{128}$ bit セキュリティの実用的なステートレス署名を提供する，SPHINCS と呼ばれる新しいハッシュベース暗号署名スキームが，2015 年に報告されている [2]。

## 6.3 Hypertrees

基本的な MSS の問題は，可能な署名の数が限られていることと，OTS 鍵ペアが merkle 木の計算前にすべて生成されていなければならないことだ。鍵生成と署名時間は木の高さ $h$ にあわせて指数関数的に増加する，つまり，256OTS 鍵ペアより大きな木は時間的にも計算的にも生成するのが高価になる。
鍵・木の生成中の計算を後回しにして，実現可能な OTS 鍵ペアの数を拡張させる戦略は，hypertree と呼ばれる merkle 木で構成される木を使用することだ。概念は，certification 木として知られる，親の merkle 木の葉のハッシュから，OTS 鍵で子のマークル・ルートを署名することだ。

最もシンプルな形 (高さ $h = 2$) では，certification 木は $2^1$ 個の OTS 鍵ペアと一緒に前計算され，最初の署名が必要なときに，新たな signature merkle 木 (signature tree 0) が計算され，certification 木の OTS 鍵ペアのうちの一つにより署名される。signature 木は $n$ 個の葉のハッシュと対応する OTS 鍵ペアで構成さ

Figure 4. linking merkle trees




れており，適宜メッセージに署名する役割を果たす。signature 木の各 OTS 鍵ペアが使用されたとき，次の signature 木 (signature tree 1) が certification 木のもう一つの OTS 鍵ペアにより署名され，署名の次のバッチが可能となる。

そのような hypertree 構造の署名 $S$ は，少し複雑になり次のものを含む。
1. signature 木から：$s$， $n$， $merkle\ path$， $root$
2. 各 certification tree から：(子のマークル・ルートの)$s$，$n$， $merkle\ path$， $root$

最初の MSS を無限に拡張するために，certification 木から下へ，木のレイヤーを入れ子にすることは理論的に可能だ。署名されるそれぞれの追加の木に対して，署名サイズは線形に増加し，その一方で hypertree 署名のキャパシティは指数関数的に増加する。

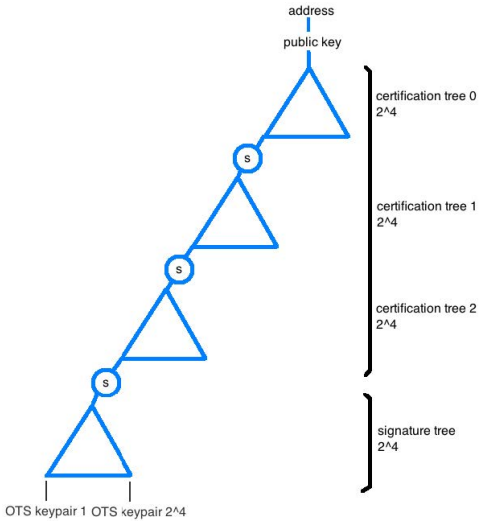
### 6.3.1 Hypertree example

hypertree 構造を用いて，どれほど容易に MSS が拡張できるか説明するために，$2^5$ 個の葉のハッシュを持つ高さ 5 の certification 木と関連する OTS 鍵ペアを考えてみる。この木のマークル・ルートは台帳アドレスを生成するために変換される。もう一つの merkle 木，つまり同一サイズ ($h_2 = 5$， $2^5$ 個の葉と OTS 鍵ペア) の signature 木を例示する。次の signature 木が生成される前に，32 個の署名が可能だ。可能な署名のトータルの数は $2^{h_1+h_2}$ であり，この場合 $2^{10} = 1024$ である。

Macbook Pro 2.7Ghz i5, 8.5GB RAM を用いて，様々なサイズに対して，OTS 鍵ペアと merkle certification 木を生成したときの結果 (最適化されていない Python コード，Winternitz OTS) は次のようになった。$2^4 = 0.5s$，$2^5 = 1.2s$，$2^6 = 3.5s$，$2^8 = 15.5s$。二つの $2^4$ 木の初期生成により構成される hypertree は，同じ署名のキャパシティで，標準的な $2^8$MSS 木に対して，$15.5s$ と比較して $1s$ 程度であった。

hypertree の深さ (あるいは高さ) を増加させても，この傾向が続く。4 つの連なる $2^4$ certification 木と $2^4$ の signature 木から構成される hypertree は，$2^{10} = 1,048,576$ 個の署名が可能でであり，署名サイズへの追加コストがあるが，それでも生成時間はたったの $2.5s$ である。

Figure 5. Hypertree construction




hypertree が対称になるための必要条件は何もないので，もし最初に二つの木から構成されていても，あとから木のレイヤーをさらに署名することによって拡張することが可能である。台帳アドレスからの署名は，それゆえ小さいものから始まり，hypertree の深さが大きくなるにつれて，次第に大きくなっていくだろう。台帳アドレスからトランザクションを生成・署名するのに merkle hypertree を使用すると，$2^{12}$ より大きいトランザクションは必要としない。それゆえに，$h = 5$ の hypertree を用いて計算的に楽に，$2^{20}$ 回安全に署名できる能力は，十分以上だ。

## 6.4 XMSS

拡張 merkle 署名スキーム (XMSS) は 2011 年に初めて Buchmann らにより報告され，昨年 IETF のドラフトとして発表された [4][7]。最低限のセキュリティ要件で選択文書攻撃を受ける条件 (PRF と第二原像攻撃) のもとで，それは，正当性証明可能なセキュアで存在的偽造が不可能なものである。そのスキームは，一味違っていて，親ノードに連結される前に子ノードに XOR ビットマスクを利用する merkle 木を介することにより，ワンタイム署名の拡張を可能にする。XOR ビットマスクにより，衝突耐性ハッシュ関数族は置き換えられる。

木の葉もまた OTS 鍵ペアのハッシュではなく，子 L-tree のルートであり，$l$ 個の葉となる OTS 公開鍵を持っている。Winternitz OTS+ は，ワンタイム署名に使用される (なお，この変種は 2011 年に初めて発表された)。

XMSS 公開鍵のビット長は $(2(H + \lceil logl \rceil) + 1)n$，XMSS 署名は長さ $(l + H)n$，XMSS 秘密署名鍵の長さは

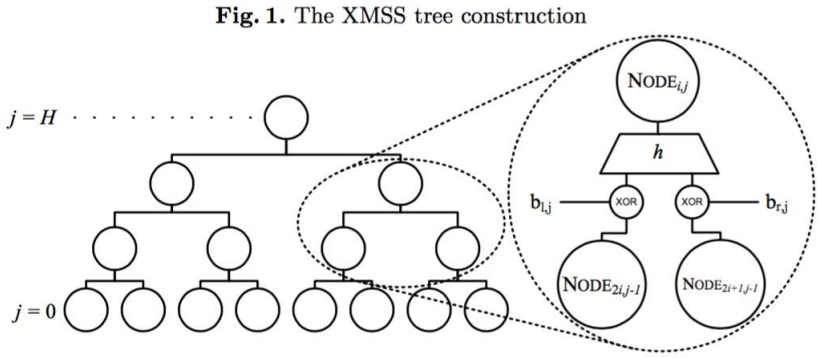
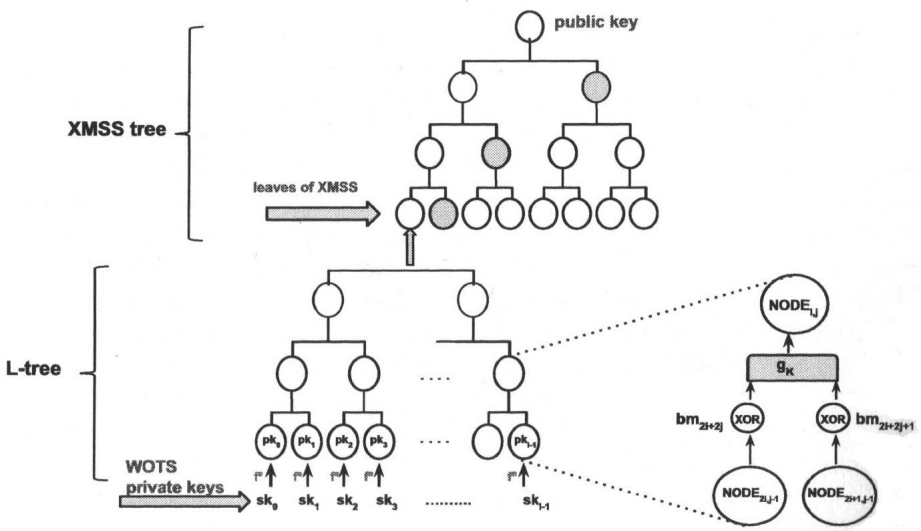



Figure.7 XMSS construction[8]




$< 2n$ である。

Buchmann は，$w = 16$ で暗号学的ハッシュ関数として 100 万程度の署名となる SHA-256($m = 256$) を使用する高さ $h = 20$ の XMSS 木に対し，Intel(R) i5 2.5 Ghz を用いて，パフォーマンスを報告している。同様のパラメータ・ハードウェアで，署名に 7ms，検証に 0.52ms，鍵生成に 466 秒かかった。それらのパラメータで獲得されるセキュリティレベルは，1.7KB の公開鍵・ 280bit の秘密鍵・ 2.8KB の署名に対し 196 ビットだった。XMSS は，鍵生成に極めて長い時間がかかるという欠点はあるが，魅力的なスキームである。

## 6.5 XMSS tree performance

すべての鍵とハッシュベースの PRF より生成されるビットマスクを持つ，4096 個の葉の XMSS 木 ($h = 12$) の構成に，QRL のテストノードには未最適化の Python ライブラリを使用すると，上述したものと同様のハードウェア (Macbook pro 2.7GHz i5，8GB RAM) で 32 秒かかった。これには，PRF を介した 8000 以上のビットマスク，300,000 を超える $sk$ の生成も含む。より効率的な merkle 木のトラバーサルアル

ゴリズムと，WOTS で $2^w - 1$ 回に対し WOTS+ では秘密鍵チェーン関数あたりたった $w - 1$ 回の演算しか要求されないということが，通常の MSS を超えた劇的なパフォーマンスの改善に貢献している。

5.75KB 程度の完全な署名が，この構成 (11.75KB の 16 進数文字エンコーディング) から得られた。これには，OTS 鍵ペア，署名，XMSS 認証ルート，OTS 公開鍵，XMSS 木公開鍵 (PRF 公開鍵 seed と XMSS 木ルートを含む) も含む。

PRF や乱数の seed を用いて生成された，さまざまな署名の量の木に対して，次のようなパフォーマンスが得られた：(h=9) 512 4.2 秒，(h=10) 1024 8.2 秒，(h=11) 2048 16.02 秒。

# 7 Proposed signature scheme

## 7.1 Security requirements

署名スキームの暗号学的セキュリティーが，今日やこの先 10 年，古典コンピューティングと量子コンピューティングによる攻撃に対して共に安全であるということが，QRL の設計の中では重要だ。$w = 16$ の SHA-256 を使用する XMSS は 196 bit のセキュリティを提供しており，力ずくの計算の攻撃に対するその安全性は 2164 年までだと予想されている [9]。

## 7.2 QRL signatures

連結された XMSS 木で構成される拡張可能でステートフルな非対称の hypertree 署名スキームを提案する。これには 2 つの利点があり，有効な署名スキームを利用することと，巨大な XMSS の構成でみられる大幅な前計算の遅れを回避する，トランザクション署名能力を台帳アドレスの生成に持たせることである。セキュリティとパフォーマンス両方の理由から，このスキームでは，W-OTS+ がハッシュベースワンタイム署名として選ばれた。

## 7.3 Hypertree construction

### 7.3.1 Key and signature sizes

hypertree 中の木の数が増えるにつれて，鍵と署名サイズは線形に増加する――しかし，署名の許容量は指数関数的に増加する。様々な XMSS 木由来の (2011 年の記述に基づく) 公開鍵と署名のサイズは，次のようになる。なお，$w = 16$，$m = 256$，$h$ は木の高さ，暗号学的ハッシュアルゴリズムは SHA-256 である。

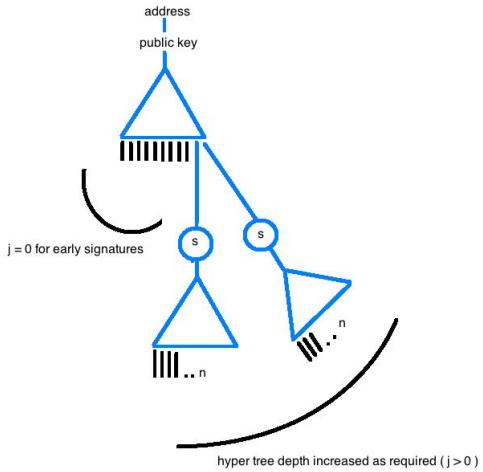
- $h = 2$，2 個の署名：公開鍵 0.59 KB，署名 2.12 KB (0.4 秒)
- $h = 5$，$2^5$ 個の署名：公開鍵 0.78 KB，署名 2.21 KB (0.6 秒)
- $h = 12$，$2^{12}$ 個の署名：公開鍵 1.23 KB，署名 2.43 KB (32 秒)
- $h = 20$，$2^{20}$ 個の署名：公開鍵 1.7 KB，署名 2.69 KB (466 秒 [3])

最終的に署名を $2^{20}$ 個生成するのに 466 秒と比較して 3 秒以内，2.69KB に対して 8.84KB の署名サイズとなる，XMSS hypertree(4 trees，$j = 3$，$h = 5$) の生成へのトレードオフは許容してもさしつかえない。

### 7.3.2 Asymmetry

非対称の木を生成することで，単一の XMSS 木の構成で早い署名が可能になる。また，その XMSS 木は，全体の署名キャパシティに犠牲を強いて，後の署名のために適宜その構成が拡張される。その理論的根拠は，ブロックチェーン台帳アプリケーションに対して大したことでないだろうことと，ウォレットが署名・鍵サイズに対する署名キャパシティのオプションをユーザへ与えるということだ。最大の木の深さ $j = 2$ で，全ての状況に対して満足するはずだ。

Figure 8. Asymmetrical hypertree




## 7.4 QRL hypertree specification

hypertree の標準的な構成に対して，次のデフォルトパラメータが適用される。

- $j = 0$ （ $j \in \{0 \leq x \leq 2\}$ ），$h = 12$ （ $h \in \{1 \leq x \leq 14\}$ ），署名数の上界 $2^{36}$，最小署名サイズ 2.21 KB，最大署名サイズ 7.65 KB

つまり，単一の XMSS 木では，$h = 12$ の 4096 個の署名が可能であり，それは適宜 $h = 14$ まで拡張されうる。ほとんどのユーザに対して，追加で木が必要になることは全然ないだろう。

### 7.4.1 Example QRL signature

最も複雑な hypertree 構成 $j = 2$，$h = 14$ を仮定すると，それぞれの XMSS 木に対する OTS 鍵ペアの位置 $n$ として，トランザクションメッセージ $m$ に対する署名は次のものを必要とする。

- Signature 木，$j = 2$ : $m$ の OTS 署名，$n$，マークル認証証明，Signature 木のマークル・ルート
- Certification 木，$j = 1$ : Signature 木 $(j = 2)$ からのマークル・ルートの OTS 署名，$n$，マークル認証証明，マークル・ルート
- Original XMSS 木，$j = 0$ : マークル・ルート $(j = 1)$ の OTS 署名，$n$，マークル認証証明，マークル・ルート

検証には，$m$ からの OTS 公開鍵生成と署名を含んでおり，与えられたマークル認証証明を確認することで Signature 木のマークル・ルートを生成する。これが次の OTS 署名に対するメッセージとなり，そこから次の OTS 公開鍵が生成され，与えられたマークル認証証明が Certification 木のマークル・ルートの再生成に使用され，それが次の Certification 木の OTS 署名に対するメッセージとなり，といった具合だ。署名は，一番高位の木，つまり Original XMSS 木 ($j = 0$) のマークル・ルートが正常に生成されたときだけ有効だ。

OTS 公開鍵は，XMSS 木署名の検証に必要ではないことに気がつく。もっとはっきり言えば，それぞれの木のマークル・ルートもまた推測可能で，それゆえに送信台帳アドレスが既知であれば，hypertree 署名検証では省略される（∵これは，QRL の署名において，最高位の XMSS Certification 木 ($j = 0$) に対するマークル・ルートから計算された派生物である———後の Accounts を参照のこと）。

署名スキームはステートフルなので，ウォレットの実装では，与えられたアドレスの hypertree にて生成される XMSS 木各々に対して，$n$ を保持・更新しなければならない。

### 7.5 PRF

seed による PRF。HMAC_DRBG。

### 7.6 Deterministic wallet

単一の SEED を用いることで，長期にわたりほとんどのユーザを満足させるはずの，かなり大きな XMSS 木が生成可能だ。セキュアなエントロピーソースは，木を生成する擬似乱数鍵を生成するためのセキュアな PRF を通ってきたこの SEED を，生成するために使用される。同様の XMSS 木を使用するひとつの欠点は，ユーザが単一のアドレスに制限されることだ（公開鍵の露見は MSS とは関係しないが）。
ビットコインやイーサリアムのアドレスは関連する公開鍵が元になり，そのようにして単一の秘密鍵・公開鍵は単一のアドレスしか生成しないだろう。しかしながら，XMSS アドレスは，マークル・ルートや公開 SEED を含む公開鍵 PK が元になっている。もし，SEED が定数のままだが木を計算する OTS 鍵ペアの数が変化するならば，そのときマークル・ルートも各々の変化に対して変化するだろう。それゆえに，1 つの OTS 鍵ペアの 1 つの加算あるいは減算ごとに，導かれるアドレスは変化する。
この特徴は，ウォレット/ノードソフトウェアによって，必要なだけ多くのアドレスの生成を可能にする，（同一の初期 SEED を使用することで適宜拡張/短縮される）XMSS 木の多くのバリエーションの生成に使用される。この情報を安全に保存するために，ステートフルで簡潔な方法は計算上些細なものである。

## 8 Cryptocurrency design parameters

ホワイトペーパーの残りで，QRL 台帳に対して提案する設計パラメータを提示する。台帳の焦点は，古典コンピューティング・量子コンピューティングの攻撃経路に対して，高いセキュリティのパブリックブロックチェーンになることだ。これは最初のドラフトであるため，どの側面ももしかすると変更される対象になる。

## 8.1 Proof-of-Stake

QRL は Proof-of-Stake アルゴリズムによって，安全なパブリックなブロックチェーン台帳になる。epoch は，10,000 block 続く。Stake validator は，一つ前の epoch の stake transaction により決定される。概念は，stake validator が各々，長さ 10,000 のハッシュの反復合成チェーンの，最後のハッシュを含むトランザクションを署名するというものだ（ハッシュ関数のセキュリティ要件を低減させるために，それぞれの反復合成で XOR ビットマスクが適用されるかもしれない）。チェーンで stake transaction が確認されると，すべてのネットワーク上のノードが，stake アドレスの暗号の同一性を次の epoch のハッシュチェーンに結びつける。

### 8.1.1 Design and randomness

すべての block に対して，現在の epoch を提供している各 validating ノードが，暗号学的に参加を証明するための，チェーンの次に続く前のハッシュを明らかにして，winning block selector になるため投票する。

HMAC_DRBG は，ブロックチェーンの SEED データトークンから 32 バイトの擬似乱数列を生成するために使用される（最初はジェネシス・ブロック，それからは，後の epoch それぞれに対して，新しい連結されたブロックヘッダハッシュより追加されたエントロピートークン）。

それゆえに，stake validator が block selector になるように選択した block はそれぞれ，そのブロックの PRF の出力に最も近い明らかにされたハッシュであることにより決定される。これは，ハッシュ・チェーンの作成時点で，staker が PRF を知ることができないため，競争するのが難しい。さらに，反復合成された（鍵付きの）ハッシュ・チェーンは，本質的に乱数列だ。最後に，stake validator が結局何らかの形で共謀しても，他の stake validator のハッシュ・チェーンの内容はまだ明らかにされていないので，それを知ることはできない。

アカウントの残高が少ないシビル・ノードあるいは stake validator アドレスからの空のブロック攻撃を軽減するため，柔軟な stake validator リストの閾値を採用する。ブロック報酬は，stake アドレスの残高に基づいて重み付けされた形で支払われる。公開されたハッシュ・メッセージでは，各ノードは，ブロックを待っているトランザクションの数と共に，トランザクション・プール内のソートされた txhash のリストの Merkle 木のルートハッシュも明かす。各ノードは上下から割合をスライスして，次のブロックで予想されるトランザクションの数を確認する。ブロックが空であるか，予想よりも少ない場合は，ブロックごとに stake contract を上昇させる（数量が乏しいものから恵まれているものまで stake validator を除外して）。block selector ノードが正当に動いている場合，また逆も然りだが，参加可能な stake validator の数は増加する。資金は，stake している epoch の間は動かないかもしれない——これは，多数の stake validator address の作成によりブロック選択を調整しようという企てを防ぐ。

## 8.2 Fees

他の台帳と比較してトランザクション・サイズが大きいほど，それぞれの取引で，取引手数料を支払う必要がある。著者は，人工的な手数料の市場（ビットコイン参照）は不要であるとの意見であり，公開されているパブリック・ブロックチェーン台帳の典型に反するように実行する。最低限の料金を支払った場合の各取

引は，他の取引と同様に有効でなければならない。マイナーが受け取ろうとする手数料は，市場により変動して設定されるべきだ。すなわち，ノード/マイナーは彼らの間で料金の下限を競争的に設定する。絶対的な最小値は，プロトコルレベルで強制される。したがって，マイナーは，自由裁量でブロックに含めるために mempool からのトランザクションを処理する。

## 8.3 Blocks

### 8.3.1 Block-times

Bitcoin にはブロック間で約 10 分の時間があるが，自然な分散によって，次のブロックが採掘されるまでにかなりの時間がかかることがある。イーサリアムのような新しい台帳のデザインはこれを改善し，セキュリティの喪失や孤立した/古いブロックの高い割合によるマイナー集中が起こることなく，はるかに短いブロック時間（15 秒）の恩恵を受けている。イーサリアムは，Greedy Heaviest Observed Subtree プロトコルの改訂版を使用している。これにより，古い/孤立したブロックをブロックチェーンに含めて報酬を得ることができる [5, 13]。

QRL は，当初より Proof-of-Stake アルゴリズムを使用する予定であるため，ブロック・タイムを 15 秒から 30 秒の間で安全に使用することが期待される。

### 8.3.2 Block-rewards

新たに作成された各ブロックには，コインベース報酬とブロック内の取引総手数料の合計に相当する報酬が入るマイニングアドレスを含む，最初の「コインベース」トランザクションが含まれている。ブロック報酬は，block selector として選択された stake validator アドレスの残高に基づいて重み付けされる。

ブロック報酬は，ブロックごとにマイニング・ノードによって再計算され，coin emission スケジュールに従う。

### 8.3.3 Blocksize

論争を避けるため，最近の $z$ ブロックの中央値 $y$ の倍数 $x$ に基づいてブロック・サイズを増加させる，Bitpay の提案からモデル化された常識にとらわれない適応型ソリューションを採用する [12]。中央値の使用は，平均ブロックサイズを変更するために，マイナーによって空白のブロックあるいは余分なブロックを含ませることを防ぐ。$x$ と $z$ は，ネットワークが従うための頑丈なコンセンサスのルールになる。

したがって，最大ブロックサイズ $b$ は，以下のように単純に計算することができる。

$$b = xy$$

## 8.4 Currency unit and denominations

QRL は，基本通貨単位として金銭トークン，すなわち *quantum*（複数形 *quanta*）を使用する。各 *quantum* は次のように最小の要素に割り切れる。

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

したがって，*quantum* の一部を含む各トランザクションは，実際には *Shor* 単位では非常に大きな整数だ。取引手数料は *Shor* 単位で支払われ，計算される。

## 8.5 Accounts

ユーザ残高はアカウントに保持される。各アカウントは，単に「Q」で始まる文字列で示される固有の再利用可能な台帳アドレスだ。

最高位の XMSS Certification 木のマークル・ルートで，SHA-256 を実行することによってアドレスが作成される。これに 4 バイトのチェックサム（マークル・ルートの SHA-256 ダブルハッシュの最初の 4 バイトから形成）が付加され，先頭に文字「Q」が付いている。すなわち，Python 疑似コードでは，次のようになる。

$$Q + sha256(merkle\_root) + sha256(sha256(merkle\_root))[:4]$$

典型的なアカウント・アドレスは，

$$Qcea29b1402248d53469e352de662923986f3a94cf0f51522bedd08fb5e64948af479$$

各アカウントには，1 つの *Shor* 単位に分割可能な *quanta* と称する残高がある。

フレッシュな OTS 鍵ペアと，各アカウントに対してそれまでに使用されたすべての公開鍵（これは，トランザクション署名とメッセージから動作中に再生成される可能性があるため，切り取られる可能性がある）を格納している QRL とを使用することで，アドレスは各トランザクションでステートフルだ。nonce と呼ばれるトランザクション・カウンタは，アカウントから送信されたトランザクションごとにインクリメントされる。これにより，ブロックチェーン全体を格納していないウォレットは，ステートフルな merkle hypertree 署名スキームでその位置を追跡することができる。

## 8.6 Coin issuance

### 8.6.1 Historical considerations

Bitcoin は最初の分散型暗号通貨であり，当初は実験的な金銭的価値がないため，通貨を完全にマイナーを通じて流通させることが適切だった。最近，Zcash はオープンソースプロジェクトへの放出初期にコインベースの採掘報酬の%で同じプロセスを選択した - その結果，信じられないほどの価格変動をもたらした。
イーサリアムなどの他の台帳では，代わりにイニシャル・コイン・オファリング（ICO）の一環として最終供給量の大部分を売却した。これは早期採用者がプロジェクトをサポートすることでまだ潜在的利益があるが，それに加えてプロジェクトそのものが開発を継続し，初期段階からプロジェクトを成長させるための資金を生

み出す可能性がある。ICO アプローチはまた，ジェネシス・ブロックから売買するために，より大きな通貨のフロートが投資家は利用可能であるため，市場が容易に発展することを可能にする。
Auroracoin（2014 年）はアイスランドのすべての人に ICO の均等な分配を提供するという異なるアプローチをとったが，開発者はコインの供給全体の 50%を自分で保有していた。
他の暗号通貨は，単純に Bitcoin を完全にクローンしたか，新しいチェーンで異なるコードベースで新たに開始した。

### 8.6.2 Interchain balance transfer

QRL のジェネシス・ブロックに挿入された現在のビットコイン台帳のステート・スナップショットに基づいて，QRL を発行することは可能だ。概念は，一意的なメッセージと署名を含むシングルユースの' インポート' トランザクションをユーザーが作成できるようにすることだ (すなわち，スナップショットの時にビットコイン残高を有するアドレスからビットコイン秘密鍵で署名された，ランダムに生成された QRL ウォレットアドレス)。この機能はある blockheight まで有効のままで，その後はコインの残りの部分が正常に採掘される。ジェネシス・ブロックの最初の膨張は，その同じ blockheight で切り取られるだろう。この欠点は，公正であればビットコイン以外の他の暗号通貨の所有者に不利益を与え，新しいユーザが実行するのが技術的にも潜在的にも努力を必要とするものであるということだ。技術的な懸念事項は，ECDSA 公開鍵を署名とメッセージだけから復元することができる可能性があることだ。これにより，プロセスで使用されるビットコインアドレスへの公開鍵が永久に公開される。これを緩和するためには，後でランダムに生成する新しいビットコインアドレスに資金を移すことが重要だ。

（イーサリアム・ホルダーのために同じ機能を許可することができる）

### 8.6.3 Proposed Issuance - draft

QRL の最初の発行は次の通りだ。

※
以降の原文パラメータはプレセール開始時点（2017/05/01）で変更されております。
https://theqrl.org/presale/presale.htmlより最新情報を赤字にて引用しますので，参照ください。

- 初期分配量は 65,000,000（*quanta*）
- 200 年後の最終分配量は 105,000,000（平滑化された指数関数的減衰曲線を供給の限界まで描く）
- プレセールの 20%（1,300,000 *quanta*）がチームによって保有され，そのうち 65%が QRL 財団に帰属する
- プレセール資金の 100%が QRL 財団によって保持され，段階的なベスティング・プログラムは研究・開発マイルストーンと結びついている

※

- ローンチ前の ICO は 1,000,000 *quanta*（最終供給量の 4.7%）
- 最初の QRL ジェネシス・ブロックを形成するために使用される 0.01 btc を超えるすべてのビットコインアドレス残高のステート・スナップショット。ビットコイン台帳から QRL 台帳にコインを 1 : 1 の割合で直接交換したい人は，ノードウォレットを介して blockheight 518400（3 〜 6 ヶ月）まで交換

ができる。

- 財団が使用するために，さらに 1,000,000 *quanta* がジェネシス・ブロック・アドレスに保持される
- 残りの供給はマイニングされる。(21,000,000 - (2,000,000 + blockheight 518400 でインポートされるビットコイン残高))

## 8.7 Coin emission schedule

ビットコインの特徴は，基盤となる金銭トークンの発行に対する希少性と上限だ。QRL は，コインの供給量に上限を $21 \times 10^6$ *quanta* に設定して，ビットコインにしたがう。コイン供給量の頑丈な天井まで，ブロック報酬の円滑な指数関数的減衰が好まれる。これは，ビットコインの「半減期」に伴う価格変動性を排除するだろう。

総供給量 $x = 21 \times 10^6$ からジェネシス・ブロックで生成した量 $y$ を差し引いた量が，$Z_0$ から指数関数的に永久に減少する。減衰曲線は，約 200 年間（2217AD まで，ブロック・タイム 15 秒で 420,480,000 ブロックまで），*quanta* が 1 つも残らなくなるまでマイニング報酬を配分するように計算される（その後のマイニングは続けることができる）。

ブロック $t$ における残りのコイン供給量 $Z_t$ は，

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

係数 $\lambda$ は，$\lambda = \frac{ln Z_0}{t}$ で計算される。ここで，t は，最後の *quanta* までの放出スケジュールにおけるブロックの総数である。518400 ブロックまでの場合，$\lambda = 3.98590885111 \times 10^{-08}$ である。ブロック報酬 $b$ は，各ブロックで次のように計算される。

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

ジェネシス・ブロックとブロック番号 518400 との間で，ビットコイン残高をインポート・トランザクションによって台帳に移すことができる。ブロック 518401 で，放出スケジュールは，新たにインポートされた残高のロックを再調整し，$Z_t$ を低減し，それに応じてブロック報酬を調整する。

# 参考文献

[1] http://oxt.me/charts.

[2] D. Bernstein. Sphincs: practical stateless hash-based signature. 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. Hülsing. W-ots+ - shorter signatures for hash-based signature schemes. 2013.

[7] A. Hülsing. 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*, page 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. 2008.