FROST threshold signature



RFC9591

Grok

ChatGPT(ちょっと怪しい) ChatGPT

簡易FROSTなるものの説明をしていたりして怪しい。Grokの方がよさそう。

FROST

Flexible Round-Optimized Schnorr Threshold Signatures

シュノア署名を基盤にした閾値署名で RFC9591で標準化されている

いわゆるマルチシグ。Bitcoinのマルチシグを効率的/柔軟にできる

Liquid でも使われている・・・というのを見かけたがそういうことはないらしい(ChatGPT)。

最終的には単一鍵でのシュノア署名に見えるので Bitcoinで使用できる

Bitcoinのマルチシグとの違い

単一鍵に集約されるので数が多くても対応できる

m-of-n にできるので、MuSig2 の n-of-n では困る場合でも使用できる

署名プロセスが2ラウンドになる

MuSig や GG18 よりも少ない

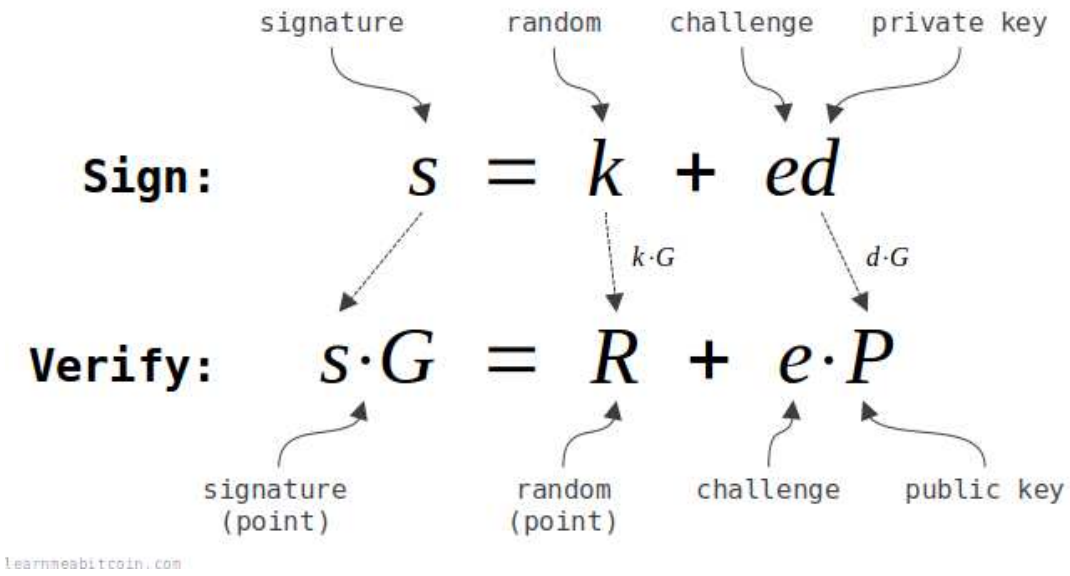
単一鍵に見えるのでプライバシーが向上する、と書いてあったが、これは TapScript でも似たようなものだろう。ただ TapScript でマルチシグすると redeem 時にわかるから、MuSig2 と同じくらいと考えればよいか。

署名を集めるプロセスで悪意がある人が止める可能性があるが、それを補強するために ROAST と併用するとよい

シュノア署名の復習

シュノア署名はこういう関係になっている。

Bitcoin ではチャレンジについて $e = hash_{BIP0340/challenge}(R.x \| P.x \| m)$ となっている。 `R.x` は、署名者は `k` を知っているので `k` から `R` を求めてx座標を取り出せば良い。検証者は署名をもらっているので、それの前半32バイトが `R.x` になる。



その他

secp256k1 では `q = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141`

分散鍵生成フェーズ(Distributed Key Generation, DKG)

1つの秘密鍵というものを作らず、各参加者がシェアを持ち公開鍵だけは全体で共有する。
DKG は 1回だけ行い、あとは署名フェーズだけを繰り返す。つまり秘密鍵は 1回だけ行い、あとはシュノア署名の random や challenge だけを変更して署名するということになる。
DKGを行うことで公開鍵が決まるので、署名フェーズだけ行ってもBitcoinアドレスは変化しない。

VSS (Verifiable Secret Sharing)の方式

- Pedersen
- Feldman
- Joint-Feldman / Joint-Pedersen
- Kate-Zaverucha-Goldberg (KZG)
- などなど.....
  - FROSTやMuSigではPedersen型が向いている
  - BLS系は KZGが効率よい
  - Feldmanは理解しやすいが隠蔽が弱め

とChatGPTにあったが、GrokではブロックチェーンはFeldmanが多いようなことを言っていた。

鍵生成

1. 各参加者 P_j はランダムな `t-1` 次多項式を作る

$$f_j(x) = a_{j,0} + a_{j,1}x + a_{j,2}x^2 + ... + a_{j,t-1}x^{t-1} \pmod q$$

定数項 $a_{j,0}$ は参加者 P_j が秘密鍵に寄与する部分とか秘密シェアの基底とか呼ばれる

この式はシャミア(Shamir)の秘密共有だと思われる。

2. シェアの配布

P_j は他の参加者 P_i に対してシェア $f_j(i)$ を個別配布する

参加者 P_i は全員分のシェアを受け取って加算: $s_i = \sum_j f_j(i)$

s_i が参加者 P_i の**秘密シェア**

各シェア $f_j(i)$ は秘密シェアを求めるために使うだけで、この次の公開鍵の構築には使わない

3. 公開鍵の構築

各 P_j は係数のコミットメント $C_{j,k} = g^{a_{j,k}}$ `(k=0,1,...,t-1)` をブロードキャストする

$C_{j,0} = g^{a_{j,0}}$ は参加者 P_j の「定数項寄与の公開値」

公開することでシェアの正当性を検証できるようになるらしい(不正防止)

全体の公開鍵は全定数項の和 $Y = g^x = g^{\sum_j a_{j,0}}$

全体の秘密鍵は $x = \sum_j a_{j,0}$

式 $Y = g^{\sum_j a_{j,0}} = \prod_j C_{j,0}$

∏ は総和ではなく総積

例えば $C_{m,0}$ と $C_{n,0}$ の積は $C_{m+n,0}$ と乗数の加算になるので、全部の積を取ると同じになる

ここまでで、秘密シェア `si` と 全体の公開鍵 `Y` が得られた。

秘密シェアを持っていると閾値 `t` の署名に参加できる。

`si` から全体の秘密鍵 `x` を復元してはならない(`t` 人未満では情報不足らしいが、a_j は展開しないのでそもそもできないということかもしれない)

署名フェーズ

ここまでで、参加者 P_i は秘密シェア s_i と全体の公開鍵 Y を知っている。

秘密シェア: $s_i = \sum_j f_j(i)$

全体の公開鍵: $Y = \prod_j C_{j,0} = g^{\sum_j a_{j,0}}$

配布

ブロードキャスト: コミットメント $C_{j,k}$

個別: シェア $f_j(i)$

ここまでにランダムで作った a はこれ以降使われることは無い。プログラムでは秘密鍵を消去するときのような念入りさで消すようにとのこと。

ラウンド1: Nonceコミットメントの配布

1. 各署名者 P_i が秘密Nonce d_i, e_i を生成(使い回さず毎回作成する)
2. コミットメント $D_i = g^{d_i}$, $E_i = g^{e_i}$ を計算
3. (i, D_i, E_i) を署名プロトコルの集約者(coordinator)に送信
4. 集約者は全署名者のコミットメントを収集し、全員に配布

   ブロードキャストでも良い。こっちの方が効率が良いというくらい。

   DKGでブロードキャストしたのは、集約者に悪意がある可能性を考慮したため。このラウンドでは乱数が一時的なものなので悪意があって失敗してもこのラウンドだけだが、鍵生成は全体に響くので。

Nonce集約 PDF 4.5章付近

1. 結合係数(binding factor) $\rho_i = H_3(i, m, \{(i, D_i, E_i)\}_{i \in S})$ を計算

   詳細な計算は RFC9591の4.4章, 4.5章参照
2. グループNonce $R = \prod_{i \in S} (D_i \cdot E_i^{\rho_i})$ を計算
3. チャレンジ $c = H(R \| Y \| m)$ を計算
4. R, m を全員に配布

   c をねつ造しないよう各自が計算できるパラメータの方を渡す

ラウンド2: 署名シェアの生成

1. 受け取った R と m を使って各自でチャレンジ値 $c = H(R \| Y \| m)$ を計算
2. 各署名者 P_i は署名シェア $z_i = d_i \cdot e_i + c \cdot \lambda \cdot s_i \pmod q$ を計算

   λ はラグランジュ係数

   ラグランジュ係数は秘密鍵の多項式分散を正しく合成するための重み

   閾値t のもとで参加者 $i \in S$ に対するラグランジュ係数は

   $$\lambda_i = \prod_{j \in S (i \neq j)} \frac{0 - j}{i - j} \pmod q$$

   参加者が `t` より多くなった場合はラグランジュ係数を調整する
3. z_i を集約者に送信

集約と完成

1. 集約者は全員の z_i を集めて $z = \sum_i z_i$

2. 最終署名: $\sigma = (R, z)$

Bitcoinスクリプトに使いたい場合、チャレンジ値 c の算出だけ BIP-340 に対応すればよいのだろうか。

メンバー変更とDKG

最初の分散鍵生成フェーズ(DKG)でBitcoinアドレスが決まるのでやり直さない限りは同じアドレスになる。

ROAST

RObust Asynchronous Schnorr Threshold signatures

署名そのものではなく、FROST のような閾値署名を補強するラッパーで Blockstream が考えた署名プロトコル(署名するプロセス)

複数の署名セッション(ラウンドと同じ?)を同時に扱うことで、悪意ある行為(遅延など)やネットワークの問題で一部が失敗しても閾値以上の正直な証明者が参加していれば署名できる