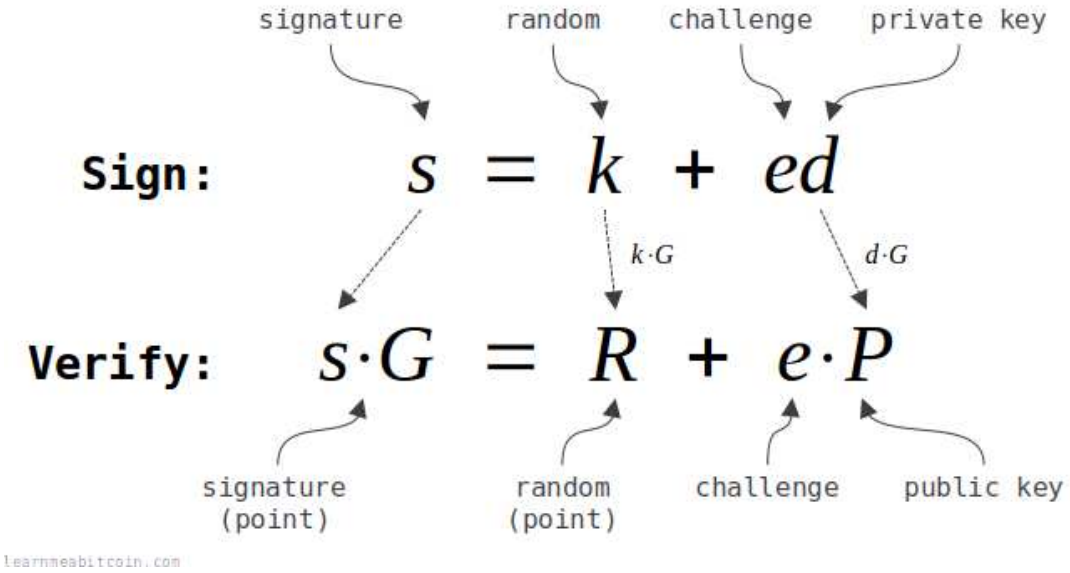
シュノア署名

BIP-340のシュノア署名。

P2TRの理解だけならシュノア署名が実装されているライブラリを使う程度でよかったのだが、 `OP_CAT` を使えるようになるとできることの理解をしようとするとシュノア署名がどういうものかがわかっている必要がありそうだった。

基本



`e` (challenge)はトランザクションデータを使ったハッシュ値で、最低でもそこについてはBIP340, BIP341を参照して同じ計算式を使うようにする。

BIP340のDefault Signing

BIP340のVerification

調査記録

Schnorr署名 —— 30年の時を超えて注目を集める電子署名 – びりあるの研究ノート

$$e = H(m; g^y p^e)$$

`m` : メッセージ

`g` : ベースポイント

`p` : 公開鍵

`y` , `e` : 署名データ

`H(x;y)` : `hash(x || y)` と同じ

ここでは `g` や `p` は整数値として扱っているようだ。
この式は `p = g^s` (BIP的な書き方だと `P = sG` か?)という関係を使ってこう表すことができる。

$$H(m; g^y (g^s)^e) = H(m; g^{y+se})$$

`r = y + se` で、 `r` は乱数。
`e = H(m || g^(y + se))` となって `e` を調整しやすい・・・らしい。
私には分からんかったが、 `(y, e)` のペアを見つけられる人は秘密鍵を知っている人なので署名として使うことができるという理屈だそうだ。

署名

1. 乱数 `r` を求める
   `g^r` は公開鍵相当
2. `e = H(m ; g^r)` を計算
   `r = y + se` なので `e = H(m ; g^(y + se)) = H(m ; g^y * g^se)`
   公開鍵を `p = g^s` とすると、 `e = H(m ; g^y * g^se) = H(m; g^y * (g^s)^e) = H(m; g^y * p^e)`
   左辺に `e` があるのに不思議な感じだが、検証に使う
3. `y = r - se` を計算
   `(y, e)` が署名である

検証

1. `p = g^s` を公開鍵とする
2. `e' = H(m ; g^y * p^e)`
   署名の `e` を求める計算式と同じ

`e' == e` となれば検証成功

Schnorr署名とOP_CATを使ったCovenants - Develop with pleasure!

`P = xG` (`x` が秘密鍵、 `P` が公開鍵)がある場合、署名は以下で求める。

1. 一時鍵 `R = kG`
   `e = int(hashBIP0340/challenge(R.x || P.x || m)) mod n`
2. `s = k + e * x`
3. `(R.x, s)` が署名

安土氏の書き方はたぶんBIP340の記号を使っている。

どちらも乱数なので `k` がびりある氏の `r` に相当すると考えて良い気がする。

`R.x` は `k` の公開鍵みたいなものだが、鍵ではないので private nonce とか public nonce とか呼ぶ？

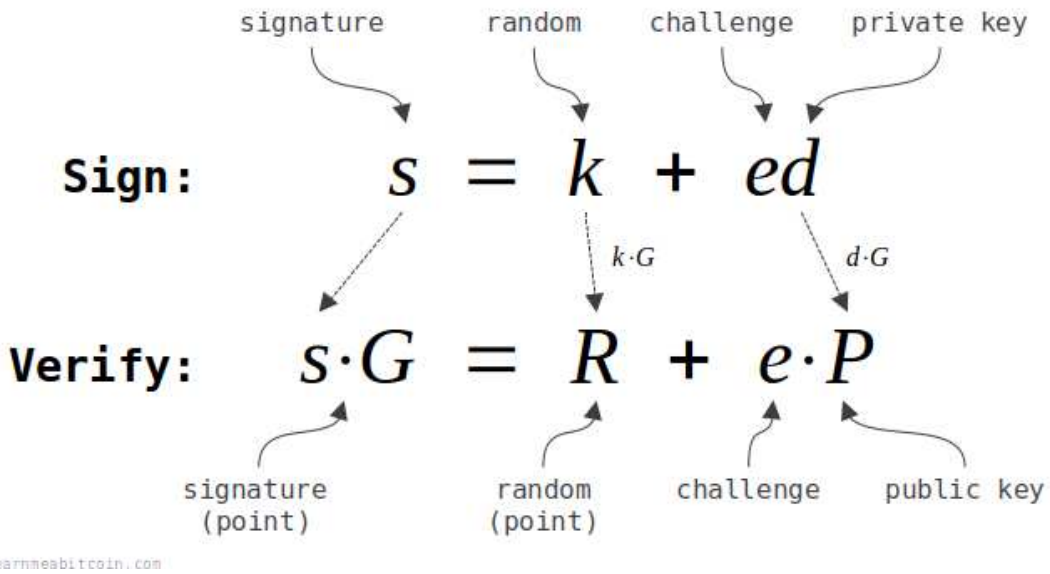
NotebookLMではこう返ってきた。

> BIP340で提案されているシュノア署名は通常64バイトで構成されます。これは、
>
> 公開ノンスに関連する点RのX座標（32バイト）と
>
> 署名スカラーs（32バイト）
>
> で構成されます。

Schnorr Signatures - Digital Signatures for Taproot

今まで式しか見ていなかったが、signとverifyはこういう関係なのか。。。

署名計算の両辺にベースポイント `G` を掛けるだけである。



BIP-340の署名データとしては `R.x` とこの図でいう `s` を連結した64バイト値を使う。

署名計算というよりは桁数が多いだけの整数演算にしか見えない。

言われてみれば、これより前に参照した解説もそういっている。challenge の `e` の求め方を説明しているかどうかくらいの違いだ。

`R.x` があるので `R.y` も導くことができる。 `s` と `P` があるので `e` さえあればverifyできる。 `m` はトランザクションデータから得られるので `e` が計算できてverifyできる。

verifyは図の式を展開した $R = s \cdot G - e \cdot P - s \cdot G + (n - e) \cdot P$ で一致することを確認すれば良い。素数位数 `n` が出てきたのは `sG - eP` という引き算になるから親切にしただけだろう。

これは全部ベースポイント `G` 上にあるから `sG - eP` を計算した X座標が `R.x` と一致すれば

良いはず。

$$e = hash_{BIP0340/challenge}(r||P.x||m) = hash_{BIP0340/challenge}(R.x||P.x||m)$$

以上を踏まえてBIP340を読み解く。

BIP340のVerification

1. 署名データ `sig` から、 `r = sig[0:32]` , `s = sig[32:64]`
   `R.x` 相当だが verify側はこの時点では `R` がわからない
2. $e = hash_{BIP0340/challenge}(r||P.x||m)$
   `P.x` は tweaked pubkey で scriptPubKey そのまま
   `m` は署名対象メッセージなのでトランザクションデータから作成する
3. `R = sG - eP`
4. `x(R) == r` となれば検証成功
   `x(R)` は `R` のX座標だから、4番で計算した結果のX座標が署名前半の `r` と一致すれば良いということか

Default Signing

verify は乱数は出てこない。challenge が sign/verify で共通なのでその求め方さえあわせておけば Default Signing の前半はあわせなくても検証できるはずだ。
BIP340だと `bytes()` がバイト列への変換だけの場合とX座標をバイト列に変換した場合の 2パターンがあって見づらいので、ここではX座標をバイト列にする方は `A.x` のように書き、それ以外は省略した。

Default Signing

```
sk = <private key>
d' = int(sk)
```
単に秘密鍵を数値化しただけ
```
P = d'G
```
公開鍵
```
d = d' or n - d'
```
`P` のY座標が奇数だったら符号反転？させる
`d'` が秘密鍵、 `P` が公開鍵でよかろう。BIP-341でいうところの"internal"である。
```
t = xor(d) & hashBIP0340/aux(a)
```
"and" が論理積のANDなのかよくわからないが、詳細は良かろう
`hashBIP0340/aux(x)` は `BIP0340/aux` での tagged hash のこと
```
rand = hashBIP0340/nonce(t || P.x || m)
```

```
k' = int(rand) mod n
R = k'G
k = k' or n - k'
```

`R` のY座標が奇数だったら符号反転？させる

`k'` が秘密鍵、 `R` が公開鍵。

```
e = int(hashBIP0340/challenge(R.x || P.x || m)) mod n
S = (k + ed) mod n
sig = R.x || S
```

署名のタイプ

ベースとなる署名とフラグの付与がある(Bitcoin Core v28.1現在)

Signature Hash Type

ベースとなる署名

`SIGHASH_DEFAULT 0x00`

トランザクションデータ全部を使って署名計算する。Taproot専用。

これ以外の署名は署名データの最後に署名タイプを1バイト付ける

`SIGHASH_ALL 0x01` (term)

トランザクションデータ全部を使って署名計算する。

`SIGHASH_NONE 0x02` (term)

INPUTだけ署名計算する。

`SIGHASH_SINGLE 0x03` (term)

INPUT全部と署名するOUTPUT indexだけ署名計算する。

フラグ

`SIGHASH_ANYONECANPAY 0x80`

このフラグがある場合、INPUTは1つだけ使う(検証するvin)。

特殊な使い方として、シュノア署名の計算方法を逆手にとった手法がある。

BitcoinのOP_CAT#e5deab