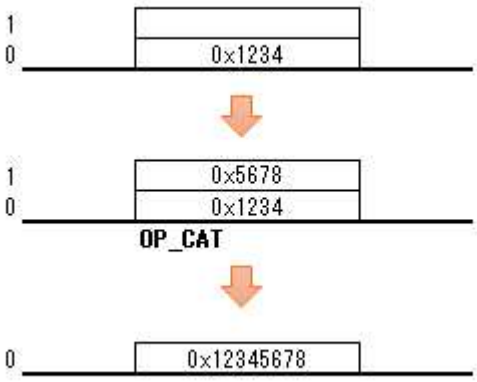
BitcoinのOP_CAT

BIP-347 bitcoin/bips
Grok
ChatGPT

結合(concatenate)の略でネコのことではない。
Linuxにも `cat` というconcatenateの略語から取ったコマンドがあるのでそこから取ったのだろう。
これを　Bitcoin に持ってくるためには Soft Fork が必要となるため、合意が取れたとしても有効になるのにそこから1～2年はかかるんじゃなかろうかと言われている。
Bitcoin’s Next Major Upgrade: A Comprehensive Assessment of OP_CAT and OP_CTV - Galaxy

基本

`OP_CAT` はBitcoinスクリプトで使用する命令で、スタック上にある2つのデータを結合して1つのスタックにする



Satoshi Nakamotoによって2010年に廃止されたそうだ(メモリを使い過ぎる危険があるため)
今回の提案は結合できるサイズの上限を小さくしている
機能はこれだけだがシュノア署名の計算方法を逆手にとっていろいろできてしまう。

当時、廃止された理由

廃止になった経緯として、データを連結するので大きいデータを簡単に作ることができてしまうという事情があったそうだ。データが大きくなるとメモリもその分必要になるのであちこちに影響が出てくるため廃止になった。

OP_CATの再導入の提案BIP-347 - Develop with pleasure!

これを復活させようとしているのがBIP-347である。"OP_CAT in Tapscript"というタイトルなので P2TR 以降で有効にしようとしているのだろう。連結後のサイズ上限を520バイトにすることでメモリの使用上限も決めている。 `OP_CAT` できるのが520バイトなのでまた新しくスタックを詰めば別の520バイトのスタックを作ることができる。それでもスタックの数自体に1000個という上限があるので最大詰めていっても520KB程度だそうだ。

昔の `OP_CAT` は `0x7e` という番号が割り振られていて、その番号は今 `OP_SUCCESS126` に当てられているものの予備として割り振っているだけなので元に戻すだけになる。

OP_CATによってできそうなこと

- OP_CAT: The Purr-fect Solution for Covenants?
  - Vaults
  - Merkle Trees for Script
  - Tree Signatures
  - Recursive Covenants
    - Starknet - Bitcoin L2ではこれを使って状態遷移ということまで提案している。

コベナンツ

- Schnorr署名とOP_CATを使ったCovenants - Develop with pleasure!
- CAT and Schnorr Tricks I - Blockstream
  - CAT and Schnorr Tricks I
- CAT and Schnorr Tricks II - Blockstream
  - CAT and Schnorr Tricks II
- Bitcoin OP_CAT Use Cases Series #1: Covenants - Implementation - Delving Bitcoin

Bitcoinのスクリプトにおける支出条件のこと。 `OP_CAT` を導入して連結された値が特定の支出条件を強制させることができることになり、すなわち特定の条件が満たされた場合にのみBitcoinが使用できるようになる。

再帰コベナンツになると擬似的にステートを持たせることができる。

Blockstream の Elements では `OP_CAT` と `OP_CHECKSIGFROMSTACK` を使うことでコベナンツの条件にもなる自分の TXID を事前にどうやってスクリプトに埋め込むのか問題を解決していたが、 `OP_CHECKSIGFROMSTACK` を使わなくてもシュノア署名の計算方法を逆手に取ることでエミュレーションできるため `OP_CAT` だけで実現できるらしい。

シュノア署名のトリック

Bitcoinスクリプトで `OP_CHECKSIG` などで署名の検証をする場合、それは「このトランザクション」に対して公開鍵と署名をもらって検証するようになっている。
Blockstream の Elements では `OP_CAT` と、任意のデータに対して署名を検証する `OP_CHECKSIGFROMSTACK` を組み合わせてコベナンツを実現していたが、 `OP_CAT` はいるが `OP_CHECKSIGFROMSTACK` なしでも機能をエミュレートさせることができる。

「シュノア署名のトリック」「シュノア署名計算の悪用」などという呼び方をしている。

これが BIP340 のシュノア署名計算

`P = xG` (`x` が秘密鍵、 `P` が公開鍵)の場合、シュノア署名は以下で求める。

1. 一時鍵 `R = kG`

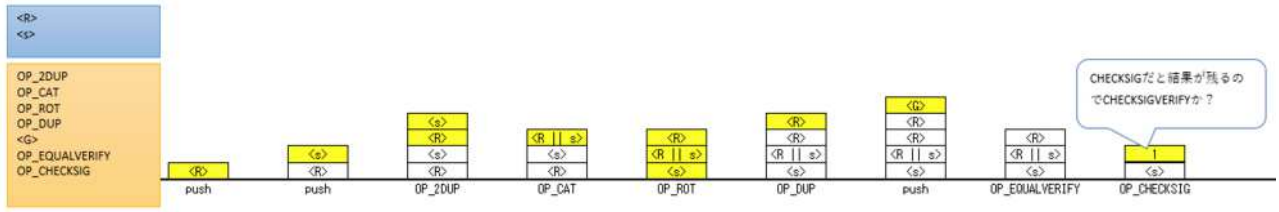
```
e = int(hashBIP0340/challenge(R.x || P.x || m)) mod n
```

2. `s = k + e * x`
3. `R.x || s` が署名

`e` の中にある `m` はBIP-341 のルールで主にトランザクションデータから作った `SigMsg` を `hashTapSighash(SigMsg)` でtagged hashした値。

説明用スクリプト

こちらはコベナンツのスクリプトではなく、その手前までの説明用スクリプトである。
最後のスタックにゴミが載っているが、説明だけなので無視して良い。
実際のスクリプトは witness stack に SigMsg の計算ができるだけのデータを載せ、SigMsg をスクリプトで計算したりするのでかなり大きくなることが予想される。



最後の `OP_CHECKSIG` が成功する場合、署名計算の `k` と `x` が `1` にするのとほぼ同等ということになる(ほぼ、なのは `+1` されるため)。
値を置き換えてみると...

`P = xG = G`

1. 一時鍵 `R = kG = G`

`e = hashBIP0340/challenge(R.x || P.x || m) mod n = hashBIP0340/challenge(G.x || G.x || m) mod n`

2. `s = k + e * x = 1 + e`
3. `R.x || s = G.x || (1 + e)` が署名

`+1` が邪魔なので、提供するのは `s` そのものではなく `e` の末尾を削ったデータにする。
そしてそのデータに 1バイトの `a` や `a + 1` を `OP_CAT` で末尾に付けることで `+1` の部分を処理する。
`OP_CAT` する値はスクリプトに埋め込むことになるので、 `e` を求める際にトランザクションの `nLockTime` をいじるなどしてハッシュ値を何度か計算し、末尾が期待する値になるよう調整する(`0xFF` だと `+1` すると 2バイト目に影響するので)。
Bitcoin OP_CAT Use Cases Series #1: Covenants - Implementation - Delving Bitcoin
署名の方が `+1` されていて、されていなければ `e` ということになる。

署名は `G` と `m` だけの値になる。
署名の前半 32バイトは `G` で、既に値が分かっている。特に必要ないのでチェックして捨てる。
後半 32バイトの `1 + e` のうち `e` はシュノア署名計算の `e` 。
その中の `m` はBIP341 のルールで作った `SigMsg` を `hashTapSighash(SigMsg)` でtagged hashした値。
`SigMsg` はトランザクションデータから作る。

何をどうするとよい？
sCrypt の実装が参考になる。といっても JavaScript 風の言語で書くライブラリなので、これをBitcoinスクリプトに落とし込むのはきつそうだ。

> Bitcoin OP_CAT Use Cases Series #1: Covenants - Implementation - Delving Bitcoin
>
> Implementation
>
> GitHub

おそらくこういった計算が必要になる。

> witness にはそのトランザクションの署名に必要なデータを与える
>
> Bitcoinスクリプトでは...
>
> witness から `e` の末尾 1バイトを削ったデータ `d`
>
> `d` に末尾1バイトを `OP_CAT` して `1 + e` を作る
>
> 末尾1バイトはハードコーディング
>
> witness の `1 + e` と計算した `1 + e` の比較

それを使った `OP_CHECKSIG`

`d` に末尾1バイトを `OP_CAT` して `e` を作る

末尾1バイトはハードコーディング

こちらは witness と比較しなくてもよいと思うが自信ない

Bitcoin スクリプトで witness のデータを使って `e` を計算

witness の `e` と計算した `e` の比較

ハードコーディングした `hashOutputs` とそれ以外の witenss のデータを使って `e` を計算

両者が一致することを確認

再帰コベナンツ

OP_CATで実現するトランザクション間の状態遷移 - Develop with pleasure!

コベナンツの実装を拡張する。

コベナンツによって output を指定できるということは、output が複数になっても同じことができるのである。

片方の output はスクリプトを、もう片方は現在の状態を持たせるようにする(たぶん `OP_RETURN`)。

witness にはコベナンツUTXO の outpoint を載せる。TXIDはコベナンツUTXO の HASH256 値だそうだ。

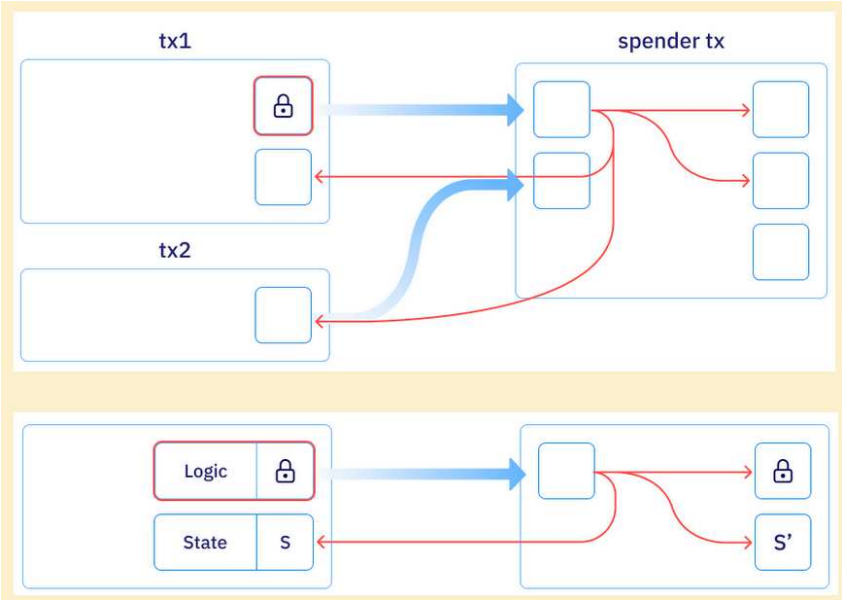
が、読んでもさっぱり分からんので元のStarkWare記事を見る。

The path to general computation on Bitcoin - StarkWare

EthereumのスマートコントラクトはわかるがBitcoinのスクリプトは知らない人に向けて書いているようだ。そしてBitcoinスクリプトでどうやるのかという詳細は書いていない。悪魔は細部に宿る(簡単そうに見えてもやるとすごく大変)そうだ。

この2つ output を持ち、片方がロジック、片方が状態とするのを名前を付けて「State Caboose パターン」と呼んでいるそうだ。caboose は貨物列車の最後に連結させる乗務員車のことだそうだ。

この caboose tx のスクリプトからは locking した状態の output も参照できるそうだ。通常ではできないから、そこがシュノア署名のトリックを使ったしくみなのだろう。



例として出てきたのは、"状態"としてカウント値を割り当てたもの。支払い側がインクリメント値を指定すると加算されていくそうだ。

Bitcoin Covenants

ここから下は未確認

擬似署名スキーム 、非両立契約(Non-equivocation contracts)

OP_CAT - Bitcoin Optech

自分の資金を危険にさらさず、競合する支払いを作成することができない曖昧な結びつき(equivocation bond)を作りたいとする。

> Aliceは秘密鍵を決定し、公開鍵を作る。
> Aliceはシュノア署名のときにやるように、ランダムな private nonce を決定し、public nonceを作る。
> locking scriptを作り、Aliceはこのスクリプトに支払う。

```
script
   <public_nonce>
   OP_CAT
   <pubkey>
   OP_CHECKSIG
```

Aliceはこのスクリプトをredeemする場合はスカラー値だけを載せるが、 `<public_nonce>` と対応するスカラー値でなければ `OP_CHECKSIG` に失敗する。

これもコベナンツの一種か？

下の句はあるが上の句がないのでスクリプトの構成がよくわからない。

使用例3

OP_CAT: A Big Step towards Bitcoin Contracts, From Vault to General Computing - Bitlayer Blog

ここは具体的なスクリプトがいろいろ書いてあった。

Schnorr Signed Trick

こちらも `k = 1` 、 `x = 1` となっている。スクリプトの形は違うがコベナンツと同じか

使用例4 merkle tree

OP_CAT Use Cases series 2: Merkle Trees - Implementation - Delving Bitcoin