Strong Federations: Thresholds and Byzantine Security

•••

Mark Friedenbach
Blockstream

Thresholds Tradeoffs

When setting up a strong federation, the choice of threshold determines the availability and security properties of the network.

- Availability: how many functionaries can be offline and the network is still operational -- blocks signed, and withdraws processed.

- Byzantine security: how many functionaries need to be dishonest or compromised for there to be a chain split.

The smallest production threshold we use at Blockstream is 8 of 11. So I will use this as an example.

Threshold Tradeoffs: 8 of 11

Availability: 8 of 11 functionaries need to be online, so the network can tolerate up to 3 functionaries being offline at any time.

If 4 or more functionaries are offline, the federation stops signing blocks / processing withdraws.

Threshold Tradeoffs: 8 of 11

Byzantine security: How many dishonest functionaries are required to split the network? Requires:

8 of 11 on side A 8 of 11 on side B

Requires 5 dishonest or compromised functionaries, when all 11 are online.

3 honest on side A 3 honest on side B

5 dishonest on both sides

Advanced Elements Script

Elementsの新機能

Mark Friedenbach
Blockstream

Feature: CHECK-SIG-FROM-STACK(スタックからの署名検証)

<sig> <data> <pubkey> CHECK-SIG-FROM-STACK

CHECK-SIGとまったく同じ動作をしますが、トランザクション自体ではなく、スタック上のメッセージに対する署名を検証します。

Elements Alpha上で "コベナンツ(約款/特約)"を実装するためにRussell O'Connorが使用しました。

https://blockstream.com/2016/11/02/covenants-in-elements-alpha.html

Feature: DETERMINISTIC-RANDOM(決定的な乱数)

<seed> <minimum> <maximum> DETERMINISTIC-RANDOM

スタックからシード、最小値、最大値を取り出します。最小値以上、最大値未満の範囲の一様分布でランダム値を計算します。同じ入力で常に同じ値を計算します。(deterministic/決定的-数学用語)

ユースケース例：確率的な支払い。アリスがボブに1円を支払う代わりに、アリスは1,000円を支払うけれども、その取引が有効になる確率を1000分の1にします。

Example: Probabilistic Payments(例: 確率的な支払い)

アリスはボブに1,000分の1の確率で有効になる1,000倍の料金の支払いをしたいと思います。通常の支払いと大体同じ方法で支払いができて、手数料が1,000分の1になるからです。.

ボブは、たとえ99.9%の有効にならない支払いのうちの1つだったとしても、0.1%は1,000倍になるチャンスがあることを知っているので、その支払いを受け入れます。ボブの得られる収入の期待値は同じですが、UTXOが細切れにならないので、ボブは0.1%で1,000倍の方を好むでしょう。

アリスはその支払いを有効になるかどうかわからないままコミットしなければなりません。

ボブはその支払いを有効にするためにトランザクションを変更できてはいけません。

Example: Probabilistic Payments

- アリスは 256-bit の数値を選んで rAlice とします。
- ボブは 256-bit の数値を選んで rBob とします。
- HASH256(rBob) を計算して hBob とします。

1. rAlice|hBob にアリスが署名したことを確認します。
2. HASH256(rBob) と hBob が等しいことを確認します。
3. random(HASH256(rAlice|rBob), 0, 1000) がゼロになることを確認します。
4. トランザクションにボブが署名したことを確認します。

Example: Probabilistic Payments

1. rAlice|hBob signed by Alice

```
sigAlice rAlice                ||  hBob CAT pubAlice CHECK-SIG-FROM-STACK-VERIFY
sigAlice rAlice hBob           ||  CAT pubAlice CHECK-SIG-FROM-STACK-VERIFY
sigAlice rAlice|hBob           ||  pubAlice CHECK-SIG-FROM-STACK-VERIFY
sigAlice rAlice|hBob pubAlice  ||  CHECK-SIG-FROM-STACK-VERIFY
                               ||
```

(continued)

Example: Probabilistic Payments

2. HASH256(rBob) equals hBob

```
rBob           ||  HASH256 hBob EQUALVERIFY
H(rBob)        ||  hBob EQUALVERIFY
H(rBob) hBob   ||  EQUALVERIFY
               ||
```

(continued)

Example: Probabilistic Payments

3. random(HASH256(rAlice|rBob), 0, 1000) is zero

```
rAlice rBob             ||  CAT HASH256 0 1000 DETERMINISTIC-RANDOM 0 1 WITHIN VERIFY
rAlice|rBob             ||  HASH256 0 1000 DETERMINISTIC-RANDOM 0 1 WITHIN VERIFY
H(rAlice|rBob)          ||  0 1000 DETERMINISTIC-RANDOM 0 1 WITHIN VERIFY
H(rAlice|rBob) 0        ||  1000 DETERMINISTIC-RANDOM 0 1 WITHIN VERIFY
H(rAlice|rBob) 0 1000   ||  DETERMINISTIC-RANDOM 0 1 WITHIN VERIFY
<random,0,1000>         ||  0 1 WITHIN VERIFY
<random,0,1000> 0       ||  1 WITHIN VERIFY
<random,0,1000> 0 1     ||  WITHIN VERIFY
<bool>                  ||  VERIFY
                        ||
```

(continued)

Example: Probabilistic Payments

4. TX signed by Bob

```
sigBob          ||  pubBob CHECK-SIG
sigBob pubBob   ||  CHECK-SIG
<bool>          ||
```

(done)

Feature: Merkle Branch Verification(マークルブランチ検証)

与えられた要素がコミットされたマークルツリーに含まれているか検証します。

```
[<leaf> <path> <root> CMB]      =>   <bool>    Check～なら真偽値
[<leaf> <path> <root> CMBV]     =>   # empty   Check～Verifyなら空
```

The inputs are:

1. (pubkey) マークルツリーのルート。

2. (witness) ツリーの要素のハッシュ値 (通常はターミナルノードまたはリーフ)。

3. (witness) ツリーを通るパスと、取得されないブランチのハッシュ値を含む 'proof'オブジェクト。

Feature: Merkle Branch Verification(マークルブランチ検証)

このopcodeは要素とそのプルーフを使用してルートハッシュ値を再計算し、最初の引数で指定された期待値と比較します。比較結果はスタックに配置される(CMB)か、VERIFYセマンティクスでは、不一致の場合にスクリプトを終了させます(CMBV)。

scriptPubKey => OVER HASH256 *root* CMBV ...
scriptWitness => ... *leaf proof*

Feature: Tail-call recursion (A better *EVAL*)(より良いEVAL)

サブスクリプトを実行できると一般的な計算が可能になります。既存のスタック操作オペコードと組み合わされると、EVAL操作を使ってさまざまなループや再帰的なサブルーチンコールを構築できます。

テールコール再帰は、EVAL操作がスクリプトの最後にあるようなものです。状態を維持する必要はありません。新しいスクリプトにジャンプするだけです。

opcodeでの実装ではなく、クリーンなスタックの意味を再定義するものです。

Feature: Tail-call recursion (A better *EVAL*)(より良いEVAL)

実行スクリプトの実行がスタックに複数の要素を残して終了する場合に、一番上の要素をスクリプトとして呼び出します。

```
Script      => [] # 空またはスクリプトの終了
Stack       => <arg1> <arg2> ... <argN> [subscript]
```

こうなります:

```
Script      => [subscript]
Stack       => <arg1> <arg2> ... <argN>
```

Example: Mathematical Challenge

triangle(5) = 5 + 4 + 3 + 2 + 1

X

X X

X X X

X X X X

X X X X X

[NB: triangle(N) = (N^2 + N) / 2]

Example: Mathematical Challenge

C:

```
unsigned int
triangle(unsigned int n)
{
     if (n == 0)
          return 0;
     else
          return n + triangle(n-1);
}
```

Haskell:

```
tri x 0  = x

tri x n  = tri (x+n) (n-1)

triangle n = tri 0 n
```

Example: Mathematical Challenge

C:

```
unsigned int
triangleX(unsigned int x,
          unsigned int n)
{
     if (n == 0)
          return x;
     else
          return triangleX(x+n, n-1);
}
```

```
unsigned int
triangle(unsigned int n)
{
     return triangleX(0, n);
}
```

Example: Mathematical Challenge

```
[tri] x 0  ==  ROT 2DROP

[tri] x 0  ||  ROT 2DROP
x 0 [tri]  ||  2DROP
X
     (done)
```

```
[tri] x n          ==  SWAP OVER ADD SWAP 1SUB 2 PICK

[tri] x n          ||  SWAP OVER ADD SWAP 1SUB 2 PICK
[tri] n x          ||  OVER ADD SWAP 1SUB 2 PICK
[tri] n x n        ||  ADD SWAP 1SUB 2 PICK
[tri] n x+n        ||  SWAP 1SUB 2 PICK
[tri] x+n n        ||  1SUB 2 PICK
[tri] x+n n-1      ||  2 PICK
[tri] x+n n-1 2    ||  PICK
[tri] x+n n-1 [tri]
     (tail-call)
```

Example: Mathematical Challenge

```
[tri] x n = DUP 0 EQUAL
            IF
                ROT 2DROP [...finally...]       # [tri] x 0
            ELSE
                SWAP OVER MUL SWAP 1SUB 2 PICK  # [tri] x n
            ENDIF

[tri] x n       ||  DUP 0 EQUAL IF ...if... ELSE ...else... ENDIF
[tri] x n 0     ||  EQUAL IF ...if... ELSE ...else... ENDIF
[tri] x <bool>  ||  IF ...if... ELSE ...else... ENDIF
```

Example: Mathematical Challenge

Contract address commits to r, and code.
to claim, provide n such that triangle(n) = r.

```
Witness: <n>
Script:  <r> SWAP
         [
             DUP 0 EQUAL
             IF
                 ROT 2DROP EQUAL
             ELSE
                 SWAP OVER ADD SWAP 1SUB 2 PICK
             ENDIF
         ]
         0 ROT 2 PICK
```

Example: Mathematical Challenge

```
n                 ||  r SWAP [tri] 0 ROT 2 PICK
n r               ||  SWAP [tri] 0 ROT 2 PICK
r n               ||  [tri] 0 ROT 2 PICK
r n [tri]         ||  0 ROT 2 PICK
r n [tri] 0       ||  ROT 2 PICK
r [tri] 0 n       ||  2 PICK
r [tri] 0 n 2     ||  PICK
r [tri] 0 n [tri]
    (tail-call)
```

Example: Mathematical Challenge

```
# triangle(100) = 5050

$ e-cli compilescript '5050 SWAP [DUP 0 EQUAL IF ROT 2DROP EQUAL ELSE SWAP OVER ADD
SWAP 1SUB 2 PICK ENDIF] 0 ROT 2 PICK'

{
  "hex": "02ba137c10760087637b6d87677c78937c8c527968007b5279",
  "length": 25,
  "asm": "5050 OP_SWAP 760087637b6d87677c78937c8c527968 0 OP_ROT 2 OP_PICK",
  "type": "nonstandard"
}
```

Example: Mathematical Challenge

```
$ e-cli decodescript 02ba137c10760087637b6d87677c78937c8c527968007b5279

{
  "asm": "5050 OP_SWAP 760087637b6d87677c78937c8c527968 0 OP_ROT 2 OP_PICK",
  "type": "nonstandard",
  "p2sh": "XMFFTyUyewNGZQtkHeoDm3iX8wQtpos9CC"
}

$ TXID=$(e-cli sendtoaddress XMFFTyUyewNGZQtkHeoDm3iX8wQtpos9CC 10)
$ echo $TXID
2081d9366e5b0b1b6a31f50ac44913c5fffc0a6be4daa568dcc09f5799a45e1b
```

Example: Mathematical Challenge

```
$ e-cli getrawtransaction $TXID true
…
"vout": [
{
      "value": 10.00000000,
      "asset": "09f663de96be771f50cab5ded00256ffe63773e2eaa9a604092951cc3d7c6621",
      "n": 0,
      "scriptPubKey": {
            "asm": "OP_HASH160 6c8f471a263085dbc972f39fe1feaf72317cf778 OP_EQUAL",
            "hex": "a9146c8f471a263085dbc972f39fe1feaf72317cf77887",
            "reqSigs": 1,
            "type": "scripthash",
            "addresses": [
                  "XMFFTyUyewNGZQtkHeoDm3iX8wQtpos9CC"
            ]
      }
},
…
$ N=0
```

Example: Mathematical Challenge

```
$ CTADDR=$(e-cli getnewaddress)
$ echo $CTADDR
CTEmjrxU2H7rcG4uWGZWZDzUPNHyGDGHEfiP27gvWfVK2jPQ5mquaqvRs1qrP4iTA5Z8K8zxa68jiR2b

$ e-cli validateaddress $CTADDR
{
    "isvalid": true,
    "scriptPubKey": "76a91425f6e4074d7d67d0466a6e1df73802386f687bed88ac",
    "confidential_key":
"024be39a5ddb0f372a8bd587b6afa43a5524984387fbe02a6a517635cb4f45aba5",
    "unconfidential": "2dctV9tKiFQL5cqy3FSQkfB3vj6WPv6fTv6",
    "ismine": true,
…
}
$ ADDR=2dctV9tKiFQL5cqy3FSQkfB3vj6WPv6fTv6
```

Example: Mathematical Challenge

```
$ RAWTX=$(e-cli createrawtransaction
'[{"txid":"'$TXID'","vout":'$N',"sequence":4294967295}]'
'{"'$ADDR'":9.9999,"fee":0.0001}')

$ e-cli compilescript '100 [5050 SWAP [DUP 0 EQUAL IF ROT 2DROP EQUAL ELSE SWAP OVER
ADD SWAP 1SUB 2 PICK ENDIF] 0 ROT 2 PICK]'
{
  "hex": "01641902ba137c10760087637b6d87677c78937c8c527968007b5279",
  "length": 28,
  "asm": "100 02ba137c10760087637b6d87677c78937c8c527968007b5279",
  "type": "nonstandard"
}
$ RAWTX=$(e-cli insertscriptsig $RAWTX 0
01641902ba137c10760087637b6d87677c78937c8c527968007b5279)
```

Example: Mathematical Challenge

```
$ e-cli sendrawtransaction $RAWTX
972c2c1de0617b99bdc74963298377ad91b274d23947c7aacc9072d183e3280b

$ e-cli gettransaction 972c2c1de0617b99bdc74963298377ad91b274d23947c7aacc9072d183e3280b
...
  "details": [
    {
      "account": "",
      "address": "2dctV9tKiFQL5cqy3FSQkfB3vj6WPv6fTv6",
      "category": "receive",
      "amount": 9.99990000,
      "asset": "09f663de96be771f50cab5ded00256ffe63773e2eaa9a604092951cc3d7c6621",
      "label": "",
      "vout": 0
    }
  ],
...
```

Feature: Bitmask signature mode(ビットマスク署名モード)

bitcoinの署名ハッシュ (sighash) モードをリコールします:

SIGHASH_ALL: 全てのアウトプットに署名します。

SIGHASH_SINGLE: インプットと同じ位置のアウトプットのみ署名します。

SIGHASH_NONE: アウトプットに署名しません。

SIGHASH_ANYONECANPAY: このインプットのみ署名します。

(default): 全てのインプットをサインする

最初の3つのモードは、ANYONECANPAYフラグと組み合わせて、合計6つの署名モードにすることができます。

Feature: Bitmask signature mode(ビットマスク署名モード)

この修正版署名パイプラインは、アトミックスワップや保証付コントラクトなどの興味深いアプリケーションを可能にします。しかし(現状のように)署名モードをいくつかに制限することは、しばしば非効率的かつ不必要な足枷となります。たとえば、次のユースケースをサポートすることは困難です：

1. 目的のアウトプットとおつり用のアドレスに署名し、他には署名しない。
2. 同じアウトプットに署名する保証付きコントラクトのすべてを持つことで、クラウドファンディングトランザクションの非対話的な構築を可能にする。
3. スマートコントラクトで複数人に支払いながら、他のアウトプットの更新を可能にする。

Feature: Bitmask signature mode(ビットマスク署名モード)

ビットマスク署名モードでは、各署名はそれぞれ2つのビットマスクを指定できます。1つはインプット用で、もう1つはアウトプット用です。

1. 各インプットに対して、対応するビットがインプット用ビットマスクに設定されている場合はそれを保持し、クリアされている場合は削除します。
2. 各アウトプットに対して、対応するビットがアウトプット用ビットマスクに設定されている場合はそれを保持し、さもなければ削除します。
3. 署名される入力は対応するビットが設定されている必要があります。

ビットマスクはwitnessの中にあり、署名によって覆われていません。このため新しいインプットやアウトプットが追加されたらビットマスクを更新できます。