マルチシグ方式のPeg-outを作る

マルチシグ方式ブリッジの試作

コントラクトへの送金がfinalizedになる

Bitcoinではconfirm数でトランザクションの安全さを図るが、Ethereumでは finalized したかどうかで判定する。

```
$ cast receipt <TX_HASH> blockNumber
$ cast block finalized --field number
```

```
$ ../target/debug/cli -c config0.toml token send
0x9fe46736679d2d9a65f0992f2272de9f3c7fa6e0
```

分散合意

Peg-Outはこういう流れになる。
1. EOAとBitcoinアドレスを登録
2. 登録したEOAがスマートコントラクトにトークンを送金
3. finalizedになったらBitcoinマルチシグウォレットから送金するPSBTの署名を集めて登録してあるBitcoinアドレスに送金

いま、PSBTの署名を集めるリーダーを分散合意で行おうとしている。
ラウンドロビンでリーダーを決め、リーダーがPSBTを作って他のノードに署名を求め、署名が集まったらトランザクションを展開する。
リーダーはタイムスロットで交代させるため、タイミングによっては2人のリーダーがPSBTを作って展開してしまう可能性がある。これは避けられない。Peg-Inのときはリクエスト内容をどのノードが作っても同じになったのでハッシュ値で「同じメッセージを処理しない」とすることで回避できたのだが、PSBTでUTXOの選択を自動にしているのもありメッセージが同じにならない。UUIDも自分で管理するためのもので共有しているものではない。

```
pegin::_P2pRequest
   struct _P2pRequest {
       btcm_addr: String,
       eoa: String,
       out_point: String,
```

```
        token_amount: u64,
    }
```

pegout::_P2pRequest

```
    struct _P2pRequest {
        psbt: String,
        tx_hash: String, // DBの主キー
    }
```

まず、PSBTはノードごとに変わる可能性があるがパラメータにする必要がある。EOAがスマートコントラクトへ送金したトランザクションハッシュは他のノードでも確認できる。共通の話題にできるのはそれくらいしかないだろう。落とし所としては、他のノードがEthereumチェーンでEOAがリクエストしたことを確認するための情報とPSBTだろう。同一メッセージかどうかはPSBT以外のデータを比較するしかない(ハッシュでも良い)。
Peg-Inの場合はそれぞれ `btcm_addr` を発行したので、それがユニークになりDB保存するkeyにもなった。Peg-Outはスマートコントラクトへ送金したトランザクションハッシュがユニークになるはずだ。UUIDは使わなくてよかったかも？

まったく同じメッセージでないなら、ハッシュでメッセージ全体をチェックしても意味が無い。

署名を集めた人しかマルチシグのウォレットを操作できないので、リクエストの送信側がチェックするのではなく受信側がチェックして署名してはいけないデータに署名しないようにするのが目的

Ethereum transaction hashが存在すること

そのEOAがPeg-Out登録されていること

Peg-Out登録でEOAとBitcoinアドレスを紐付けているが、悪意がある人にBitcoinアドレスを変更される可能性がある。スマートコントラクトでBurnする関数を作ってBitcoinアドレスをDBではなくチェーンに保存する方が安全か。

PSBTの送金先アドレスと送金額が期待と一致すること

PSBTのお釣りアドレスがマルチシグウォレットになっていること

お釣りアドレスは存在しない可能性がある

つまりVOUTは1つか2つ

`OP_RETURN` に transaction hash を入れてもよいかもしれない

PSBTに署名できるかどうかはBDKに任せておけば良い？

偽のPeg-Outを行う目的は、実際に自分が持っていないトークンをBurnしたと見せかけることでマルチシグウォレットから自分に対してBitcoinを送金させることだろう。

なので「Burnしたと見せかける」についても気を配る必要がある。

コントラクトに送金した、という点はだますことができない。チェーンを確認すれば分かる。

そのEOAもだますことはできない。金額も同様。

その送金が既にPeg-Out済みという可能性はある

トランザクションが自分のノードが運用し始めるより前であれば判断できないので署名すべきではない。

現在からあまりにも古いトランザクションであっても除外すべきだろう。基本的にはfinalized になった直後のみとしたい。せいぜい±1ブロック程度か。

あるいはPeg-Out処理中という可能性もある

先に到着したリクエストだけ処理する

先と後のどちらが正解なのかはわからないので気にしても仕方ないだろう。

であれば、ハッシュ値でメッセージをチェックするのはあまり意味が無いが、Peg-Inでは同じメッセージかどうかチェックできるので、一時的なチェック程度にやっておこう。

第2案

Ethereumでスマートコントラクトに送金したtransaction hashに変更した。

とりあえずチェーンから情報を取ってくることはできるようになった。

```
_P2pRequest
```

```
struct _P2pRequest {
    tx_hash: String,
    psbt: String,
}
```

これだけだと `tx_hash` から送金元EOA、 `psbt` から送金先Bitcoinアドレスと金額程度は取れるが、それ以上は無理。

だがこれ以上に渡せるデータも思いつかない。

それとは別に、今から処理する予定の Peg-Out データをDBに登録するタイミングについて悩んでいる。

Peg-Inの場合はユーザ登録したマルチシグBitcoinアドレスを先にノード間で共有し、それに対する入金をブロックチェーンでチェックし、最初に見つけたリーダーノードが処理開始する。

リーダーの署名要求を受け取ったときには既にDBには何らかの登録がある。もし署名要求を受け取ったとき既に処理済みであれば無視することができる。

Peg-OutはEOAとBitcoinアドレスの登録は先に行うが、実際に処理するPeg-Outはそれとは別のDBで管理する。リーダーの署名要求を受け取ったときにまだDBにその情報を持っていないこと

があるのだ。
いろいろ考えたが、リーダーからの署名要求が来たときにはブロックチェーンを見ないといけないので、DB登録がなければ自分で確認しにいくように作るしかないか。

第3案

タイムスロット方式で自分がリーダーになってEthereumのイベントを前回から今回まで調べて各イベントについてPeg-Out処理をする。それはよいのだが件数が増えると1回で全イベントを処理するには時間が足りないことがある。そのときどうするか。

- 今の作りではイベントを通知する処理とPeg-Outの処理は別になっている。
  - イベントを通知するだけのところは時間がかからないので件数が多くても問題ない
  - 時間がかかるのはPeg-Out処理で、BTCマルチシグウォレットから送金するPSBTを作って他のノードに署名要求を送信しDB状態を `Pegouting` にする。中断するならこちら。
    - 署名要求を受信したノードはPSBT署名をしてDB状態を `Finalizing` にし署名応答を返す。
    - 署名応答を受け取ったリーダーはPSBTを結合してトランザクション展開しDB状態を `Finalizing` にする。
    - 展開したトランザクションがconfirmしたのを確認したら各自のDB状態を `Done` にする。
  - イベントはブロックごとなので一度通知するとそのブロックまでは処理済み扱いになってしまい次回は通知されない
    - が、それは自ノードだけなので、次ノードは自分が処理していない分のイベントを処理し始め、既に署名要求を受信して処理しているならスルー、未登録ならPSBT署名をして同じように処理する。
- この方式はうまくいきそうだが、ラウンドロビンで一周するまでにブロックのイベントを全部処理できなかったら忘れられてしまうという問題がある。
  - そこで、イベントの通知だけでDB状態を `Registered` に変更し、Peg-Out処理するのはDB状態が `Registered` のものだけにするという方法を考えた。
    - イベント通知をリーダースロットで行う必要はないだろう。が、別のスロットで行うと他ノードのPSBT署名要求が来るかもしれない。