Mastering Bitcoin

Chapter 7: Advanced Transactions and Scripting

kekeho

2022/06/13 Bcali

Introduction

本章の内容

- 6章では, 基本的なスクリプトであるPay-to-Public-Key-Hash (P2PKH)について紹介した
- 本章では, より高度なスクリプトを解説
  - **Multisignature**: Unlockの際に複数人の署名が必要
  - **Pay-to-Script-Hash (P2SH)**: Multisignatureの改良版
  - **Data Recording Output (RETURN)**: チェーンにデータを書き込める
  - **Timelocks**: 指定された時間までUnlockできない
  - **Segregated Witness (Segwit)**:

Multisignature

概要

- **M-of-N**方式: N個の公開鍵がスクリプトに記録され, そのうちM個以上の署名が提供されないとUnlockできない (N ≤ 3, N ≥ M)
- **OP_CHECKMULTISIG**が用意されている
- Locking Script
  - `M <Public Key 1> <Public Key 2> … <Public Key N> N CHECKMULTISIG`
- Unlocking Script
  - `<Signature 1> <Signature 2> … <Signature M>`

Multisignature

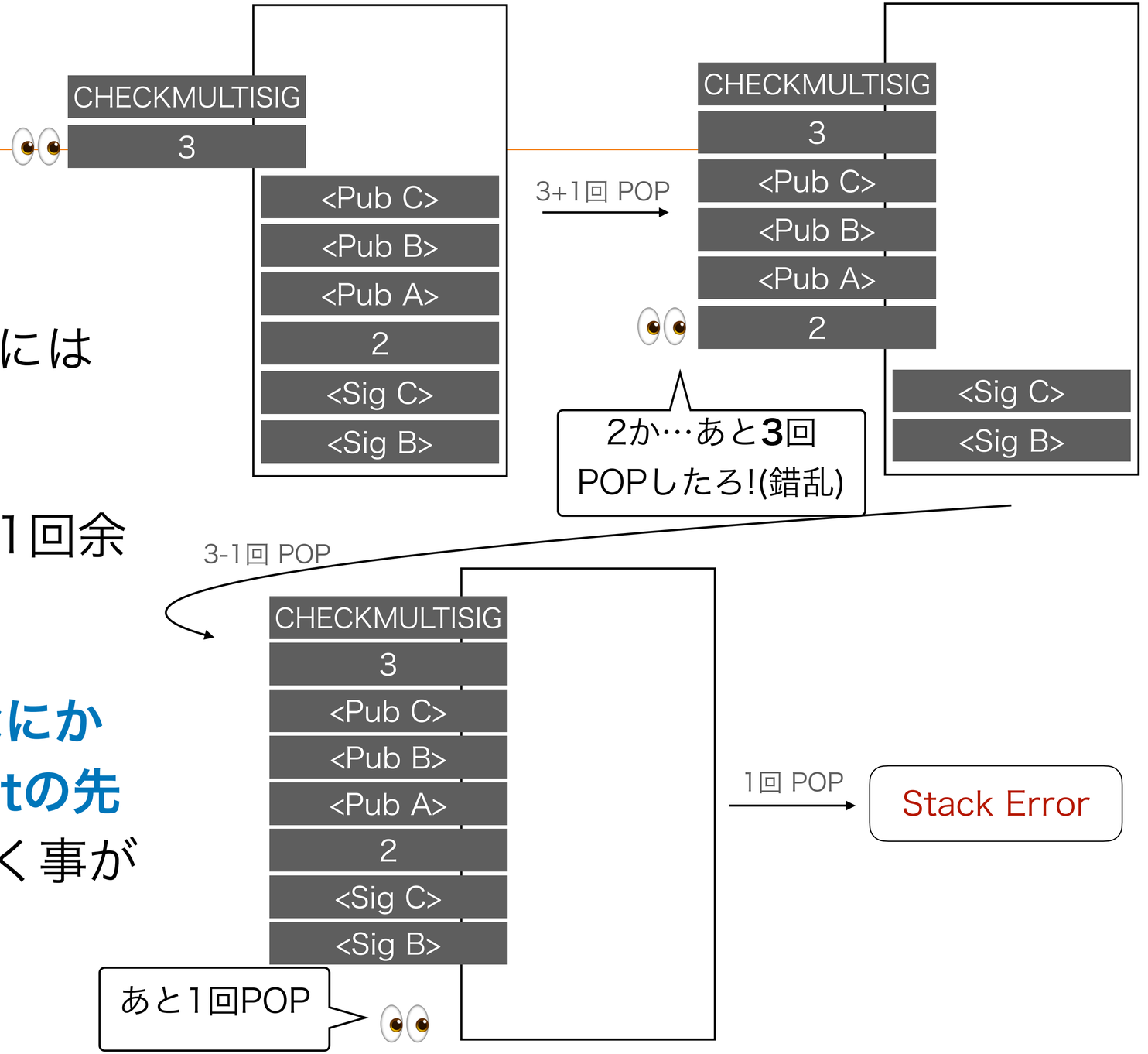
例: 2-of-3

- Locking Script
  - `2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG`
- Unlocking Script (example)
  - `<Signature B> <Signature C>`
- Unlocking Script + Locking Script
  - `<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG`

Multisignature

Bug

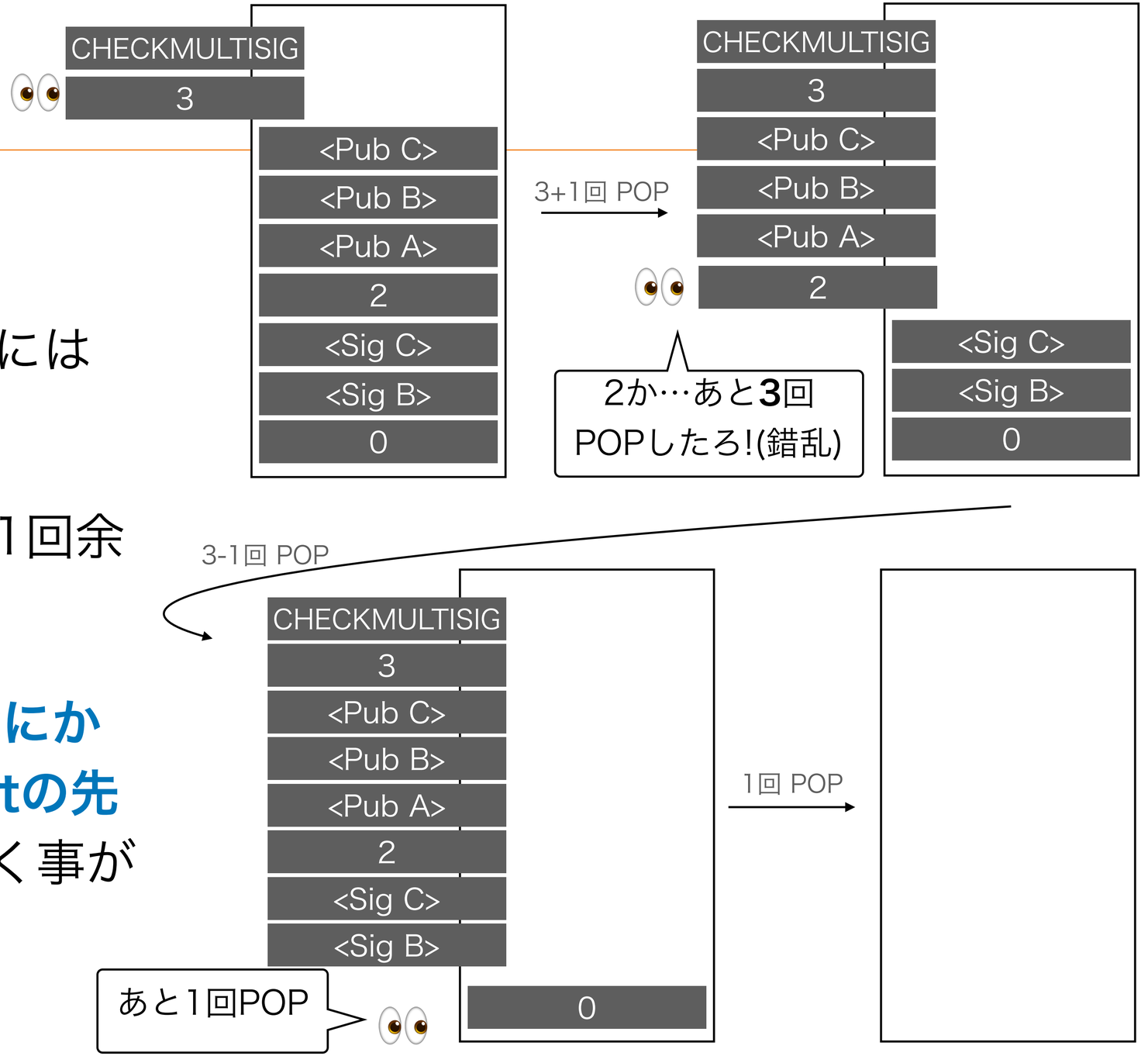
- OP_CHECKMULTISIGの実装にはバグがある
  - N+M+2回POPするはずが, 1回余分にPOPしてしまう
- 余分な値は無視されるので, **なにか適当な値をUnlocking Scriptの先頭に入れておく** (0を入れておく事が多い)



Multisignature

Bug

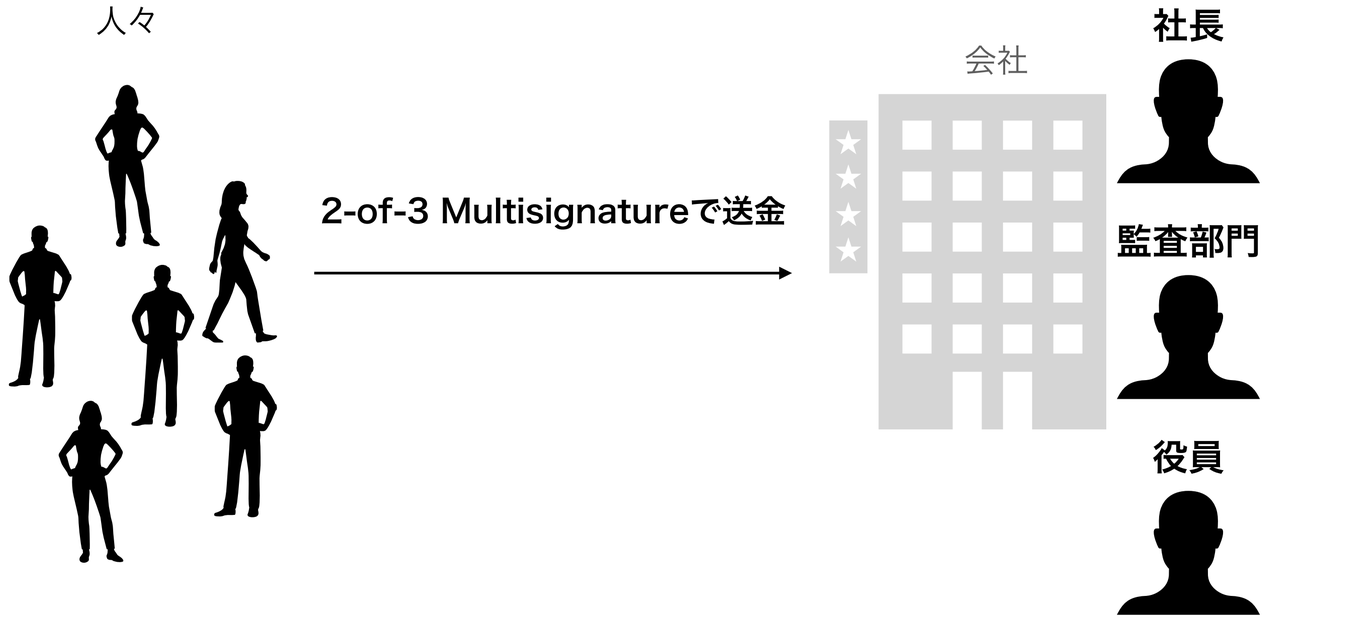
- OP_CHECKMULTISIGの実装にはバグがある
  - N+M+2回POPするはずが, 1回余分にPOPしてしまう
- 余分な値は無視されるので, **なにか適当な値をUnlocking Scriptの先頭に入れておく** (0を入れておく事が多い)



Multisignature

ユースケース

- 会社のお金を, 複数人での合意なしに引き出せないようにする
- コーポレートガバナンスの向上



過半数が賛成しないと使用できないように

Pay-to-Script-Hash (P2SH)

Multisignatureの課題

- Multisignatureは, Locking Scriptが複雑
- Locking Scriptのデータサイズが大きい
  - 送金する側に手数料の負担がのしかかる
  - フルノードにとっても, 大きなサイズのUTXOを保持するのは負担

Pay-to-Script-Hash (P2SH)

概要

- Redeem Script:
  - `2 <Public Key 1> <Public Key 2> <Public Key 3> <Public Key 4> <Public Key 5> 5 CHECKMULTISIG`
- Locking Script
  - `HASH160 <20-byte hash of redeem script> EQUAL`
    - 20byteのハッシュ値: `RIPEMD160(SHA256(redeem script))`
- Unlocking Script
  - `0 <Signature 1> <Signature 2> <redeem script>`

Pay-to-Script-Hash (P2SH)

検証の流れ

1. Redeem scriptがハッシュ値と等しくなるか検証
   - `<redeem script> HASH160 <redeem script> EQUAL`
2. Unlocking scriptでredeem scriptを解除
   - `0 <Signature 1> <Signature 2> 2 <Public Key 1> <Public Key 2> <Public Key 3> <Public Key 4> <Public Key 5> 5 CHECKMULTISIG`

Pay-to-Script-Hash (P2SH)

P2SH Address

- **P2SH Address**: Redeem scriptの20byte Hashを, Base58Checkでエンコードしたもの
- Prefixとして5を付けてエンコードすると, 先頭が3になるので, ひと目でP2SH Addressとわかる
- 会社は, P2SHアドレスを顧客に教えるだけで済む
- 顧客のウォレットは先頭が3のアドレスを見たら, 以下のスクリプトでLockするだけ?
  - `HASH160 <decodeBase58Check(P2SH Address)> EQUAL`

Pay-to-Script-Hash (P2SH)

P2SHの利点

- 複雑なScriptを短いものに置き換え, トランザクションサイズを小さくできる
- スクリプトはアドレスとして使える
- スクリプト構築の負担を, 送信者ではなく受信者にシフトできる
- フルノードがUTXOの複雑なScript(=大きなデータサイズ)を保持しなくていい
- 長いスクリプトのデータ保存における負担を, 支払い時から使用時(将来)にシフトさせる
- 長いスクリプトの高い取引低数量のコストを送信者から受信者にシフトする

Data Recording Output (RETURN)

導入の背景

- 送金に使うだけじゃなく, チェーンにデータを記録したい
  - ビットコインの分散型・タイムスタンプ型台帳の活用
    - ある時刻におけるファイルの存在証明…など
- 宛先アドレスを自由に使い, データを記録
  - 結果的に使用できないUTXOを生成することになる
- ブロックチェーンの肥大化に繋がると反対する人も

Data Recording Output (RETURN)

概要

- RETURNオペコードの導入
  - `RETURN <data>`
  - 80byte分, チェーン上にデータを書き込める
- このトランザクションは, UTXOとして保持する必要はない

Timelocks

概要

- ある時点以降の支払いしか認めないトランザクション
- 今までも, トランザクションのnLocktimeフィールドで指定ができたが…
- UTXOレベルでタイムロック・オペコードが導入された(2015〜16年)
  - `CHECKLOCKTIMEVERIFY`
  - `CHECKSEQUENCEVERIFY`

Timelocks

nLocktimeの限界

- nLocktime: トランザクションのヘッダのLocktimeフィールドで, ブロック高 or Unix Timestampを用いて使用可能なタイミングを指定
- TXをコピーして, nLocktimeフィールドを書き換えれば有効にできてしまう問題があった

Timelocks

Check Lock Time Verify (CLTV)

- Outputのredeem scriptに`CHECKLOCKTIMEVERIFY`オペコードを追加
  - 有効でない場合, 実行に失敗する
- P2PKHスクリプトにCLTVを適用する例:
  - 元: `DUP HASH160 <Public Key Hash> EQUALVERIFY CHECKSIG`
  - 適用後: **`<Block height or Unix Timestamp> CHECKLOCKTIMEVERIFY DROP`** `DUP HASH160 <Public Key Hash> EQUALVERIFY CHECKSIG`
    - 有効な場合, 時間パラメータがスタックの一番上に残るのでDROPを付ける

Timelocks

Relative Timelock: nSequence

- nLocktimeとCLTVは, 時間を絶対指定していた
- Relative Timelock: UTXOがチェーンに記録されてからの時間を指定できる
  - nSequence: トランザクションのフィールドで指定
    - 例: マイニングされてから少なくとも30ブロック経過したときに有効

Timelocks

Relative Timelock: CHECKSEQUENCEVERIFY (CSV)

- Relative TimelockをScriptで実現する: CHECKSEQUENCEVERIFYオペコード
- CLTVと同じような形で使える

Timelocks

Median-Time-Past

- それぞれのノードは違う時間で動いている
- Median-Time-Past: 直近11ブロックのタイムスタンプを取り, その中央値を取ることでコンセンサスタイムを得る
  - これをTimelockの計算に用いる
    - (現実世界の)現在時刻と約1時間ほどずれることに注意

Scripts with Flow Control

Bitcoin Scriptにおける条件分岐

- Bitcoin Scriptでは条件分岐を行うことができ, 複雑なScriptを構築できる

擬似コード

```
if (condition):
  code to run when condition is true
else:
  code to run when condition is false
code to run in either case
```

Bitcoin Script

```
condition
IF
  code to run when condition is true
ELSE
  code to run when condition is false
ENDIF
code to run in either case
```

Scripts with Flow Control

○○VERIFY

- VERIFYが付くオペコードは, 特定の条件が成り立たなければその場で実行を停止し, 成立すれば実行を継続する
  - Pythonのassertに似ている
- 例:
  - ```
    <Bob's Sig> <hash pre-image> HASH160 <expected hash> EQUALVERIFY <Bob's Pubkey> CHECKSIG
    ```
    ```

Scripts with Flow Control

VERIFY or IF

- 同じ処理でも, VERIFYを用いたほうがオペコードを削減できる

VERIFY

```
HASH160 <expected hash> EQUALVERIFY <Bob's Pubkey> CHECKSIG
```

IF

```
HASH160 <expected hash> EQUAL
IF
   <Bob's Pubkey> CHECKSIG
ENDIF
```

- ELSE説を含む場合にIFを, そうでなければVERIFYを