Fix the past and build the future, in a sensible way.

感应合约简介和基本原理

Sensible Contract Introduction

Gu Lu, 2021.04

Bitcoin SV 1st Bootcamp

SensibleSV

SATOPLAY
SATOPLAY.COM

Bitcoin SV
BOOTCAMP

Sensible Contract in a nut shell

Fix the past = 溯源

Build the future = 协作

Fix the past and build the future, in a sensible way.

预备 (Prerequisities)

分步 (Step by Step)

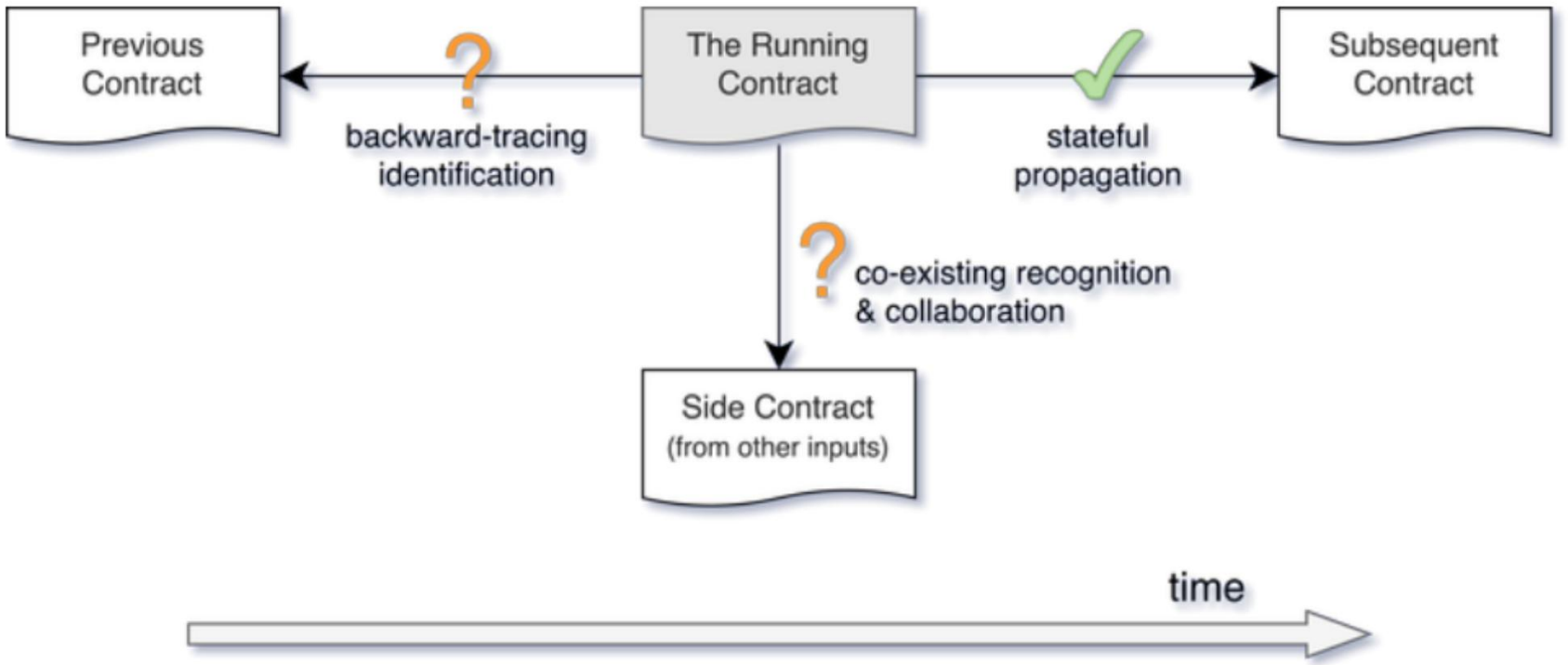
释疑 (Questions)

Brief History

Bitcoin Script	→	programmable
sCrypt	→	High-level
PUSH_TX	→	TX Data accessing
Stateful Contract	→	State propagating

Limitations

1. 我要往哪里去？ （如何对后继合约施加约束）
2. 我是谁/有谁同我一起？ （如何获知当前合约与哪些合约共存）
3. 我从哪里来？ （如何识别当前合约的出处）



为什么需要溯源?

因为需要解决合法性问题（防伪）

比特币价值单位的本质

“聪”是一个非常特殊的价值单位（**独立存在，脚本无关**）

- 每一聪，都以 utxo 的形式存在于系统内，天然是一种独占资源
- 其独占性和合法性天然由节点软件行为保证，这是**系统的义务**

- 拥有解锁的能力即拥有 utxo，你花了，别人就花不了（**排他性，独占性**）
- 不管写出多么漏洞百出的比特币脚本，都无法通过编程来增加新的 satoshi ，复制或销毁已有的 satoshi（**数量恒定，脚本无关**）

Token 价值单位的本质

- 通过脚本创建出的资源，正常情况下，不具有前述数量恒定性和代码无关性。
- （主观上）假设你在脚本中计算错了，不小心在某个关键业务点多加了一次，那 token 数量可能就 double 了。
- （客观上）其他人可以伪造一笔看起来正常的 token，扰乱已有的系统的业务逻辑。

结论：因为 token 不是比特币系统的“一类公民”，所以我们才需要手动的去确保它的合法性。

BTP 如何解决溯源问题

- 基于 oracle 的 BTP 方案，就是通过 oracle 这样一个独立于区块链的外部机制，来维护这种合法性。

- 通过维护一个 utxo 集，来保证能以最低成本最快速地知道一个 utxo 是否合法。

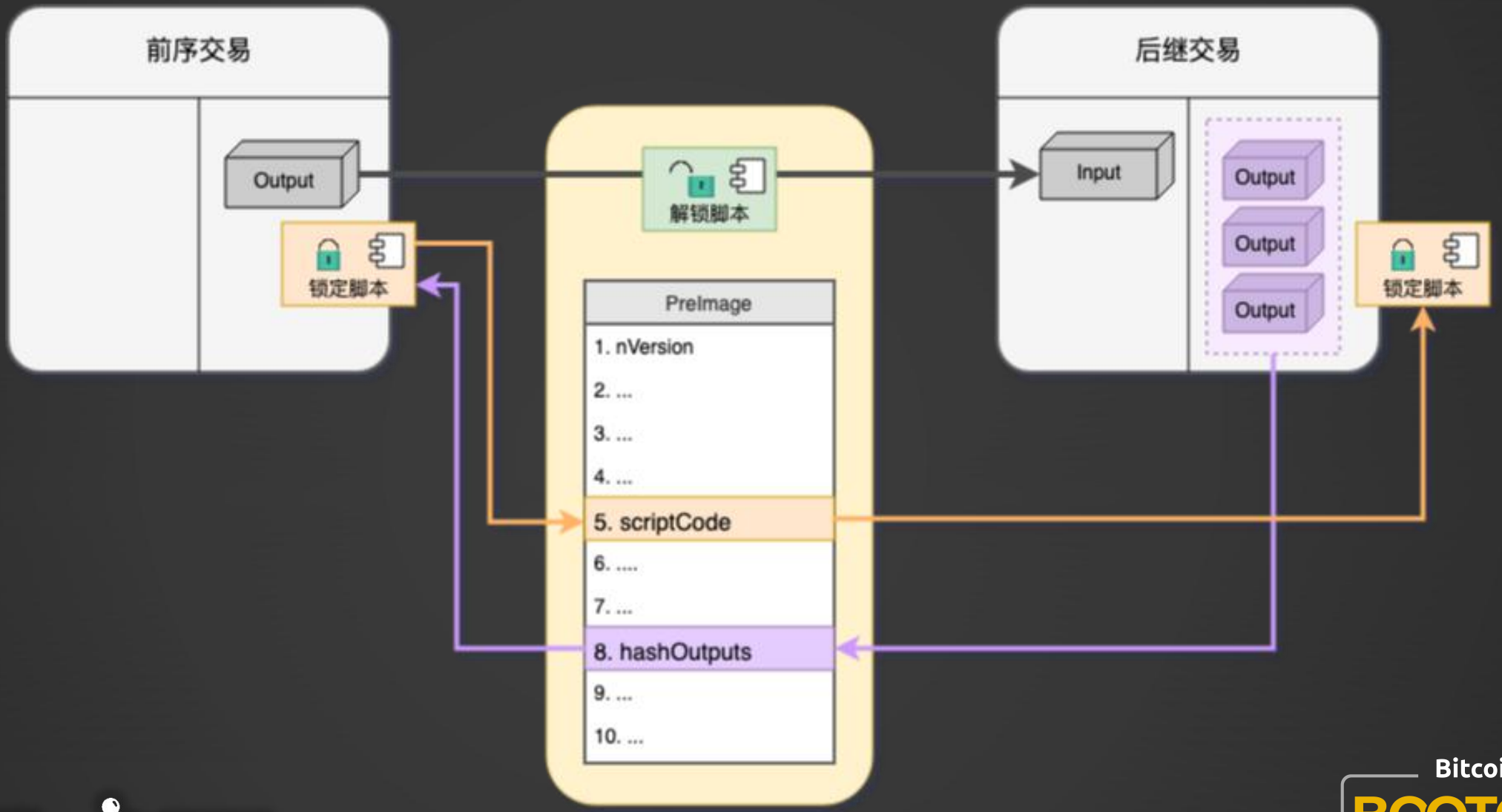
- （工程问题）一旦有了链外的状态，就需要维护这个状态，确保它的可用性及可靠性，涉及到状态更新，状态恢复，状态膨胀，等等问题；也需要多个 token server 来避免单点。

PUSH_TX

- 使得包含了众多关键字段的，当前交易的 PreImage 可以在解锁时被传入，并可通过 OP_CHECKSIG 被证明是真实的。

1. nVersion of the transaction (4-byte little endian)
2. hashPrevouts (32-byte hash)
3. hashSequence (32-byte hash)
4. outpoint (32-byte hash + 4-byte little endian)
5. scriptCode of the input (serialized as scripts inside CTxOuts)
6. value of the output spent by this input (8-byte little endian)
7. nSequence of the input (4-byte little endian)
8. hashOutputs (32-byte hash)
9. nLocktime of the transaction (4-byte little endian)
10. sighash type of the signature (4-byte little endian)

Stateful Contract



准备知识回顾小结

- 比特币脚本的几个重要的演化节点
- 比特币合约的限制（功能边界）
- 为什么合约需要溯源?
- 一个典型的 oracle 方案 (BTP) 是如何解决溯源的
- PUSH_TX & Stateful Contract

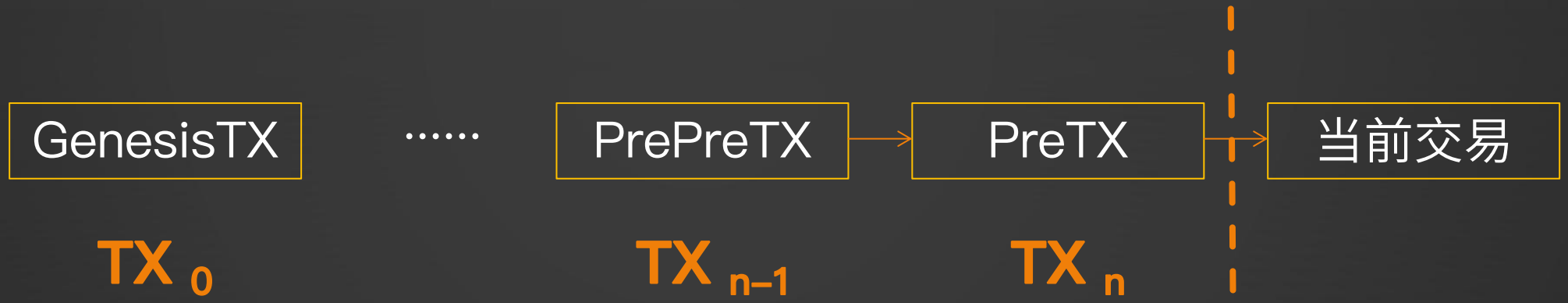
预备 (Prerequisities)

分步 (Step by Step)

释疑 (Questions)

0. 初始约定

- 当前正在解锁的 UTXO 所在的交易 **PreTX**，也可看作 TX_n
- PreTX 的前置交易 **PrePreTX**，也可看作 TX_{n-1}
- 第一笔源头交易，也就是根交易 **GenesisTX**，也可看作 TX_0



1. 简单回溯

Stateful Contract 可以保证交易的延续，可是关键的问题在于是否有能力**向前回溯**。

也就是：确认前序交易的关联性。

简单的做法是：

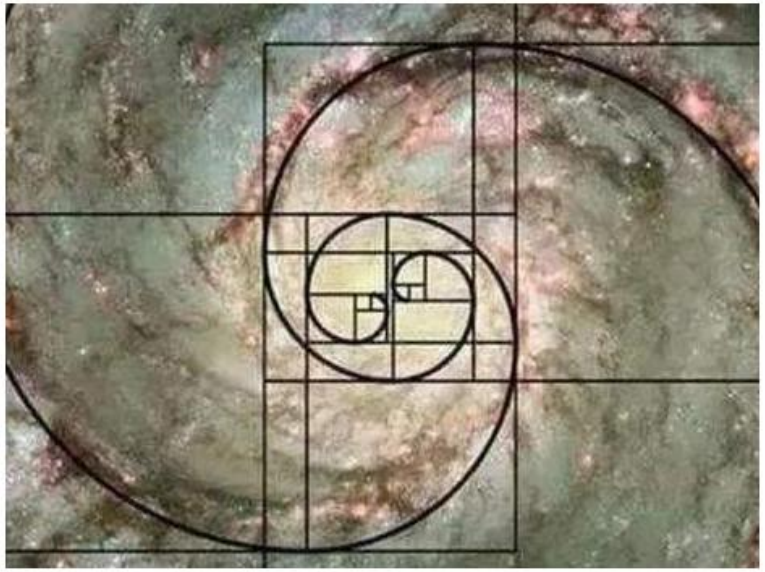
把 PrePreTX / PreTX 整体放入解锁脚本中，就可以提取关键信息来确认其相关性。

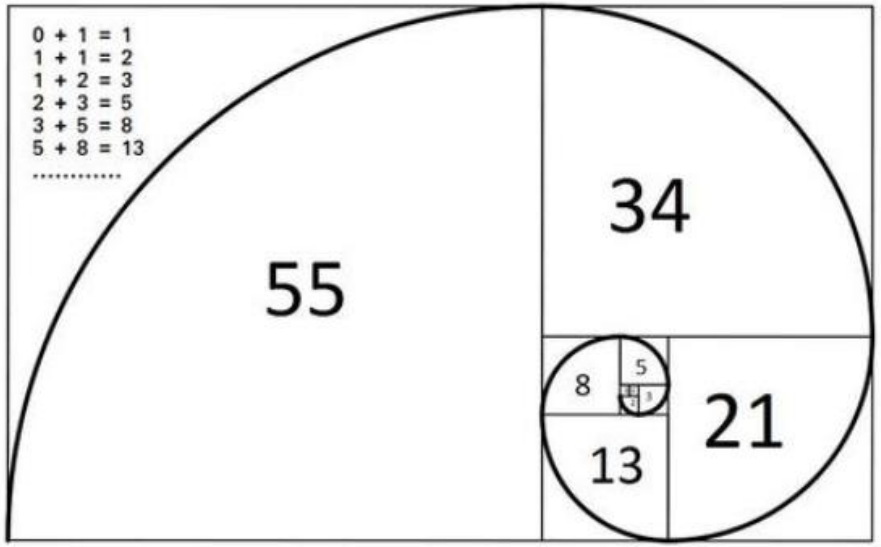
但是，这会导致膨胀。

2. 膨胀原因分析

斐波那契膨胀 $Size_{T+1} = Size_{T} + Size_{T-1}$

高速膨胀，近似 2^n





2. 膨胀原因分析

那么如何避免这种膨胀呢?

膨胀的内容都在解锁脚本里面，**不能放解锁脚本!**

不放入整个交易，而是只放入**锁定脚本**。

锁定脚本能够被用来识别脚本行为。

因为锁定脚本是**定长**，所以不会导致持续的膨胀。

将锁定脚本作为“合约指纹”入栈，这样就够了吗？

攻击者完全可以提供一个长得一模一样的锁定脚本

但来自于一个**系统外部的无关**交易

怎么办?

3. 签名验证锁定脚本

rabin 签名

签名

(合约外)

验签

(合约内)

确认相关性

签名 对 **PrePreTX 的锁定脚本**，及**相关上下文信息**， 和 **PreTX 的 TXID** 一起签名。

2. 对 u4 + "花费该 u4 的 Tx" 做签名（可将该签名称为 sig_spent_u4）：

```
sig(txid, index, value, scriptCode, spendByTxID)    # sig_spent_u4
```

验签 在解锁时**在合约内验签**，确认 **PrePreTX 和 PreTX 的相关性**。

```
// verify rabin signature
int h = Util.fromLEUnsigned(this.hash(msg + padding));
require((sig * sig) % this.rabinPubKey == h % this.rabinPubKey);
```

4. 解锁流程

1）压入 PreImage

当前正要解锁的 UTXO 的锁定脚本 L_n

2）压入 锁定脚本

前序交易的锁定脚本 L_{n-1}

3）验签

rabin(sig, pub) == rabin(msg, pub)

$L_{n-1} = L_n$

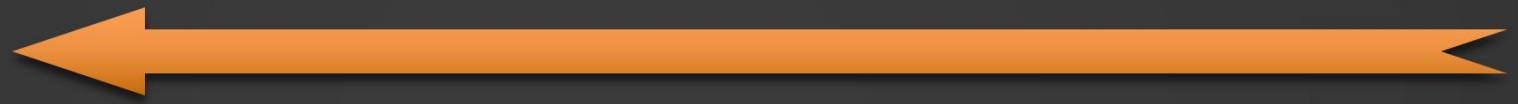
请注意，

锁定脚本完全一致，并不是必要条件。

① $L_{n-1} = L_n$

② rabin verified

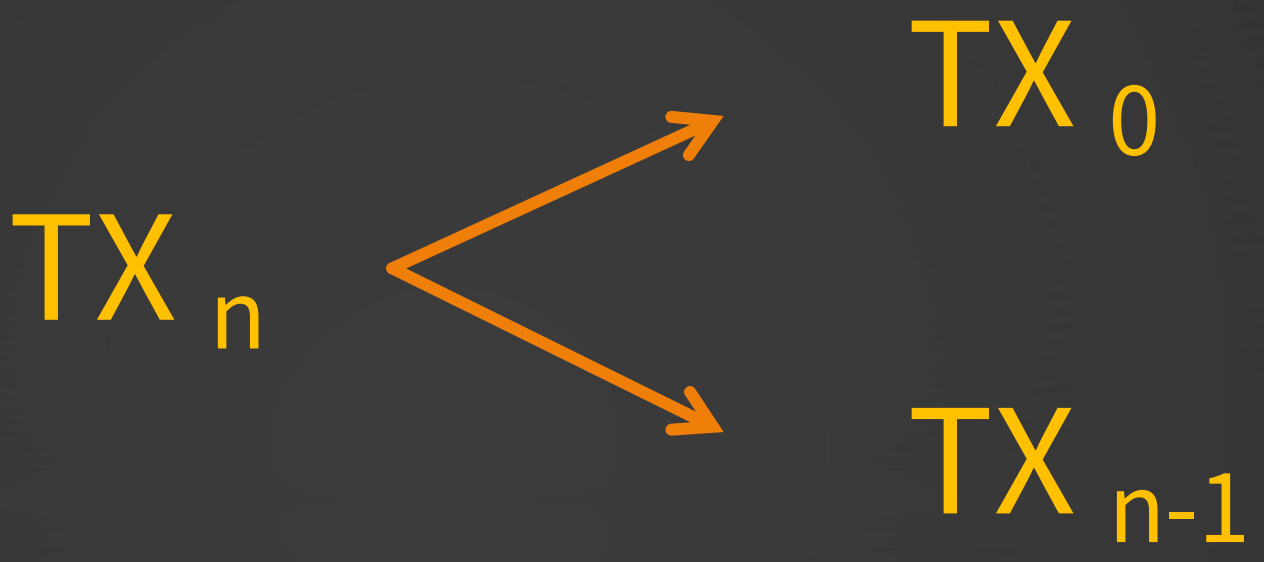
$$TX_1 \ldots\ldots TX_{n-2} \leftarrow TX_{n-1} \leftarrow TX_n$$



$$TX_0 \leftarrow TX_1$$

匹配 Genesis TX 标记

对于任意给定的一笔合约交易 TX_n



数学归纳法

5. 化简后的逻辑

任何一笔有效合约交易 TX_n，其合法性：

- ✓ 要么来源于前一笔有效合约交易 PreTX (也就是 TX_{n-1})
- ✓ 要么来源于创世交易 GenesisTX (也就是 TX_0)

合法性传导

6. 优化合约尺寸

在解锁 TX_n 时，需要传入两次锁定脚本

- 其中一次是 PreImage 的第五项 TX_n 的锁定脚本 L_n
- 另一次是前序交易 TX_{n-1} 的锁定脚本 L_{n-1}

对于合法交易，这两份锁定脚本是完全一样的。也就是说，如果锁定脚本编译出来是 5KB，那解锁脚本至少会包含 10 KB。

6. 优化合约尺寸 (去重)

- PreImage 内的锁定脚本 L_n
- ~~前序交易的锁定脚本 L_{n-1}~~

（这里改为传前序交易的锁定脚本哈希 **hash(L_{n-1})**）

在合约里直接比较 **hash(L_n)** 和 **hash(L_{n-1})** 即可

hash160 生成格式与地址一致，可以在合约协作时把脚本hash与地址同等对待。

提纲

预备 (Prerequisities)

分步 (Step by Step)

释疑 (Questions)

释疑 1. 为何需要一个额外的签名器?

能否把计算过程放入合约，这样不就消除了这个（看起来多余的）签名器?

执行计算需要源数据，签名器的目的，就是为了把源数据从链上推到链下，避免数据随着交易越堆越多。

释疑 2. 为何升级 PreImage 就不需要签名器?

因为节点软件 (bitcoind) 有足够的上下文来计算我们所需要的数据，所以应直接在节点内完成计算，并把数据放在新增的字段里。

释疑 3. 升级 PreImage 是改协议吗?

这正是此提案的精髓之处。在（感应合约）之前，对基础协议的修改被认为是必须的，但实际上只需要在特定的地方处理下，即可保证修改被局限在比特币节点软件实现层面。在 2016 年才经由 Johnson Lau 和 Pieter Wuille 编写的 BIP 143 而引入比特币节点软件实现的 PreImage 结构，是后期引入的概念，我们的观点是，PreImage 并非基础比特币协议的一部分，而只是其软件实现。

释疑 4. 真的无法伪造吗?

我们来伪造一把试试就知道了。

假设一个输出 $Output_{fake}$ 内包含了一个看起来一样的锁定脚本，$TX_{n'}$ 在溯源时就会失败。因为溯源需要提供一个合法的 $TX_{n'-1}$。

- 如果能提供一个真实有效的 $TX_{n'-1}$，就是真实有效的前序交易。
- 如果提供不了，说明这个输出 $Output_{fake}$ 是伪造的，无法被解锁。

释疑 5. 签名器作恶有何办法?

1. 支持多签（如 10 / 20）
2. 可以通过昨天蒋杰提到的版本号机制更新列表，并保持有效集

Sensible Tech

1. full-featured contract logic
   - no stateful oracle
   - no 3rd-party authentication
2. full-featured contract data payload
   - verifiable critical fields
   - no op-return for this
3. fully decentralized
   - miner validation
   - no need for authenticator / validator
4. “ bitcoinic ”
   - suite well with metanet
   - support SPV exactly as same as bitcoin

Sensible Tech (comes at a price)

1. full–featured contract logic
2. full–featured contract data payload
3. fully decentralized
4. “bitcoinic”

comes at a price:

1. a minimal signature service (external)
2. heavy–weighted scripting (bigger size)

BCPs

BCP 01

NFT

BCP 02

Token (FT)

BCP 03

Unique Contract

BCP 0x ...

(to be revealed)

"automation of agreements with easily definable transaction steps"

– the "official narratives" about contract

Thank you

https://sensiblecontract.org

Gu Lu
Sensible Contract