How to solve “back to genesis” problem[1-2] for L1 Token Contract in BSV Blockchain

Author: ZWS

[1] https://xiaohuiliu.medium.com/peer-to-peer-tokens-6508986d9593
[2] https://medium.com/@buildonbsv/back-to-genesis-simplest-explanation-7a9264ca6aed

问题之背景：preimage构成（版本1）

The proposed digest algorithm computes the double SHA256 of the serialization of:

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)

问题之背景：preimage构成 (版本2)

Field	Data	Size	Description
Version	01000000	4 bytes	Which version of transaction data structure we're using. → 1. nVersion
Input Count	01	Variable	Indicates the upcoming number of inputs.
Input(s)			
Output Count	01	Variable	Indicates the upcoming number of outputs.
Output(s)			
Locktime	00000000	4 bytes	Set a minimum block height or Unix time that this transaction can be included in. → 9. nLocktime

Input(s):

Field	Data	Size	Description
TXID	796…efc	32 bytes	Refer to an existing transaction.
VOUT	01000000	4 bytes	Select one of its outputs.
Key Size	6a	Variable	Indicates the upcoming size of the unlocking code.
Key	473…825		A script that unlocks the input.
Sequence	ffffffff	4 bytes	→ 7. Sequence

Output(s):

Field	Data	Size	Description
Amount	4baf210000000000	8 bytes	The value of the output in satoshis.
Lock Size	19	Variable	Indicates the upcoming size of the locking code.
Lock	76a9…88ac		A script that locks the output.

2. hashPrevouts

3. hashSequence

8. hashOutputs

4. outpoint → the UTXO

5. scriptCode = Lock Script

balance

6. value spent

tail of signature

10. signhash

Tx 的数据结构以及与 Preimage 数据中10个分量的对应关系

L1膨胀问题的根源：having to verify a transaction’s parents’ parents

实际上的需求是，要将一个交易A的输出的数据 A.OutputXX 或Hash值Hash(A.OutputXX)传到其子交易B的子交易C的输入解锁脚本(C.InputX)中，并保持这个过程的数据的不可篡改性。

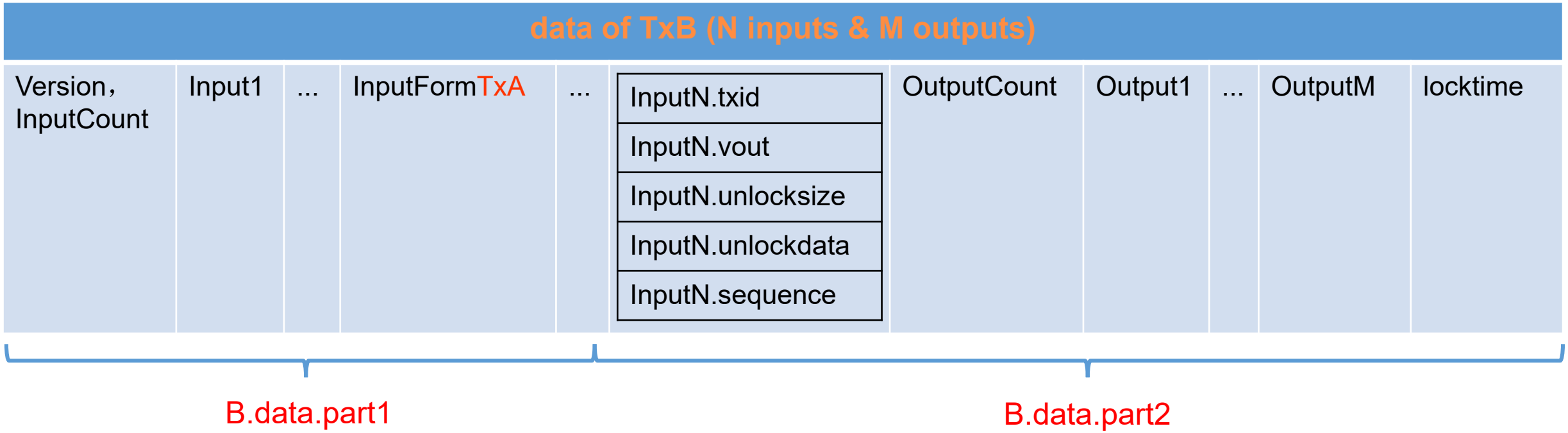
解决要点：

1、OP_PUSH_TX所需要构建的preimage并不需要用当前交易的所有数据来计算得到，其可以由其组成的10个确定有限长度的数据来构建出preimage，这些数据可以被唯一的由Hash锁来固定，其中最重要的是hashPrevouts数据，用它可以使得当前正在验证的交易输入的解锁数据包括当前交易数据的所有的输入的TXID&VOUT，并不包括其他可以引发数据大小膨胀的数据。

2、利用partialHash技术，可以只通过输入中间的B交易的前一部分（B.data.part1）的hash值以及后半部分数据（B.data.part2），来确定B.data.part1的正确性，即其确实是B交易的raw数据的后一部分。

L1 Token膨胀问题的解决方法的核心：构建一个合适B交易（假设Tx链为: TxA -> TxB -> TxC）

交易B的数据如下所示，数据将分成两个部分：B.data.part1 and B.data.part2



需要：

- B.data.part1数据的大小为512bit的整数倍；B.data.part2 数据包括InputN数据，根据前一部分B.data.part1数据的大小限制，也可能含有前一个交易Input(N-1)的有限大小的数据，以保证含有InputN数据的hash chunks的完整性。

- InputN.unlockdata中的数据包括 TxB 数据的所有inputs的txid&vout，这个数据可信度由InputN交易输入所花费的输出InputN.txid.getOutputLockscript(InputN.vout)为应用OP_PUSH_TX技术的解锁脚本并编写了对应的检查代码来保证。

解决verify a transaction’s parents’ parents时数据膨胀的方法流程

1. 对TxC的某个的输入（TxC.InputX）的验证，其应用了OP_PUSH_TX技术，相应脚本利用preimage(TxC)编写了代码可以获取数据：TxC.InputX.txid=TXID(TxB)，根据获取的TXID(TxB)，通过TXID(TxB)=Hash(partialHash(B.data.part1)||B.data.part2)来验证输入的partialHash(B.data.part1)与B.data.part2数据的正确性。

2. 根据得到的验证正确的B.data.part2数据，得到验证正确的TxB.InputN.txid与TxB.InputN.vout，标记这两个数据所指向的交易为TxD，通过TxB.InputN.txid=Hash（partialHash（TxD.frontpart）|| TxD.lastpart）来获取验证正确的 TxD.lastpart ，并验证TxB.InputN.txid与TxB.InputN.vout指向的Lockscript确实应用了OP_PUSH_TX技术，其确实编写了对应的检查代码来保证“InputN.unlockdata中的数据包括 TxB 数据的所有inputs的txid&vout”

3. 根据TxB的B.data.part2数据中的可信的InputN.unlockdata数据，获得TxB 数据的所有inputs的txid&vout，即得到TxA数据的TXID，在这过程中使用给的 partialHash(B.data.part1)、B.data.part2、partialHash（TxD.frontpart）、TxD.lastpart等数据均具有确定有限的大小。

4. 根据获得的TXID(TxA)，可以通过TXID(TxA)=Hash(partialHash（TxA.frontpart）|| TxA.lastpart)来获取TxA中的所有输出数据，并可以验证其中的LockScript的内容符合L1 Token等Contract合约的要求。确保完成了“back to genesis”所需要的 “verify a transaction’s parents’ parents”

如何防止TxB最后一个交易输出使用一个伪造的OpReturn数据，来代替TxB.data.part2？

方法1（待完善）：

1. 在前面的过程中，拿到TxB.InputN.txid（= TXID of TxD）后，根据 pow(TXID of TxD -> merklePathofTxD -> BlockHeader)，来保证这个TxD确实在链上，保证Lockscript of TxB.InputN 得到了矿工的执行。

2. 根据TXID of TxD得到TxD.lastpart后，在验证 Lockscript of TxB.InputN 确实是编写了OP_PUSH_TX验证代码时，该验证代码不仅检验了“InputN.unlockdata中的数据包括TxB 数据的所有inputs的txid&vout”，还将检验了最后一个交易输出的内容，确保其数据末端不包括用于伪造的OpReturn类数据。

如何防止TxB最后一个交易输出使用一个伪造的OpReturn数据，来代替TxB.data.part2？

方法2：

改变Raw Tx Data 的格式，要求其最后一个输出的Lockscript数据后面放置一个LockscirptSize数据。

方法3：

在preimage中加入当前解锁的utxo所在的交易的输出的总个数 (number of B.output)，并要求TxC.InputX.vout = number of B.output，再通过preimage数据中的scriptCode数据来确定其不是用于伪造的OpRetrun交易数据。