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

[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

交易版本 (4个字节)

hashPrevouts (输入outpoint的哈希32字节哈希)

hashSequence (输入sequence的哈希32字节哈希)

当前输入outpoint (32字节txid + 4字节位置)

当前输入的锁定脚本（ Varint格式)

从当前输入所花费的satoshi(8字节)

当前输入的 nSequence (4字节)

hashOutputs (输出的satoshi+输出脚本组合的32字节哈希)

交易的nLocktime (4字节)

签名的类型(4字节)

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) 展开：

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) 展开：

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

4. outpoint → the UTXO

5. scriptCode = Lock Script

6. value spent

tail of signature

8. hashOutputs

10. signhash

Tx 的数据结构以及与 Preimage 数据中10个分量的对应关系

问题之背景：preimage构成，版本3

1. nVersion 为交易的版本号（比特币为交易格式的升级预留了版本号位）。
2. hashPrevouts 一般为所有输入 UTXO 的位置指针的序列化数据的 双SHA256 Hash 值。即为：SHA256^2(Serialize(**TxPre1 TXID** + **TxPre1_VOUT** + **TxPre2_TXID** + **TxPre2_VOUT** + ...))，其中 **TxPre1_TXID** 为花费的第一个 UTXO 所在的 Tx 的 TXID，**TxPre1_VOUT** 是一个数值，指出要花费的第一个 UTXO 在其 Tx 的 Outputs 中的排序位置（序号）。**TxPre1_TXID** 与 **TxPre1_VOUT** 共同给出了 UTXO1 的全局指针（此处感谢 aaron）。当 TxNow 花费的 UTXO 只有一个时，不存在 **TxPre2** 等。 在少数交易中，其收到 SIGHASH_ANYONECANPAY 的非标准使用而有变化。
3. hashSequence 为所有输入的 nSequence 的序列化数据的 双 SHA256 Hash 值：SHA256^2(Serialize(**TxNow_Input1_nSequence** + **TxNow_Input2_nSequence** + ...))。使用 nSequence 可实现交易在交易池中等待更新、延迟上链的功能。 在少数交易中，其收到 SIGHASH_ANYONECANPAY 的非标准使用而有变化。
4. outpoint 指当前要进行 OP_CHECKSIG 验证的 UTXO 的位置指针的序列化数据，即 Serialize(**TxPreCurrentUnlock_TXID** + **TxPreCurrentUnlock_VOUT**)。因每一个 UTXO 花费时都需要做 OP_CHECKSIG 验证，于是对应每一个 OP_CHECKSIG 操作，都会有一个依赖 **当前需解锁 UTXO** 的不同的 Preimage 。与前述 hashPrevouts 是局部与整体的关系。
5. scriptCode of the input 是当前要打开（执行）的脚本锁，即 **Lock_In_TxPreCurrentUnlock。**某些时候，其因 OP_CODESEPARATOR 的加入而有所变化。
6. value of the output spent by this input: value of the amount of bitcoin spent in this input, 当前要花费的 UTXO 的比特币余额。
7. nSequence of the input： 此项表示当前进行解锁 Unlock 验证的 Input 所对应的 nSequence，可表示为 **TxNow -> InputCurrentUnlock -> nSequence**，与前述 hashSequence 是局部与整体的关系。
8. hashOutputs 一般为当前交易的所有输出的 余额+脚本锁的字节长度+脚本锁 的序列化数据的 双 SHA256 Hash 值，即 SHA256^2(Serialize(**TxNow OUT1 Amount** + **TxNow OUT1 LockSize** + **TxNow OUT1 Lock** + **TxNow OUT2_Amount** + **TxNow_OUT2_LockSize** + **TxNow_OUT2_Lock** + ...))。 某些时候，其会因 sighash 使用了非常用值而有所变化。
9. nLocktime 设定了最小的 block height or Unix time，交易只有在此之后才能被上链。
10. sighash type of the signature： 顾名思义，其表示签名的类型，会在 OP_CHECKSIG 要求输入的签名数据中被给出。其值通常为 ALL，表示对所有输入与所有输出签名。其他取值会导致前述的某些参数的涵义有所变化，本文中不考虑此种情况。

总结一下，下图中给出了在一个 transaction data 中，其中的各部分数据与 Preimage 中的数据的对应关系：

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.out1或Hash值hash(A.out1)传到其子交易B的子交易C的输入解锁脚本(C.inx)中，并保持这个过程的不可篡改性。

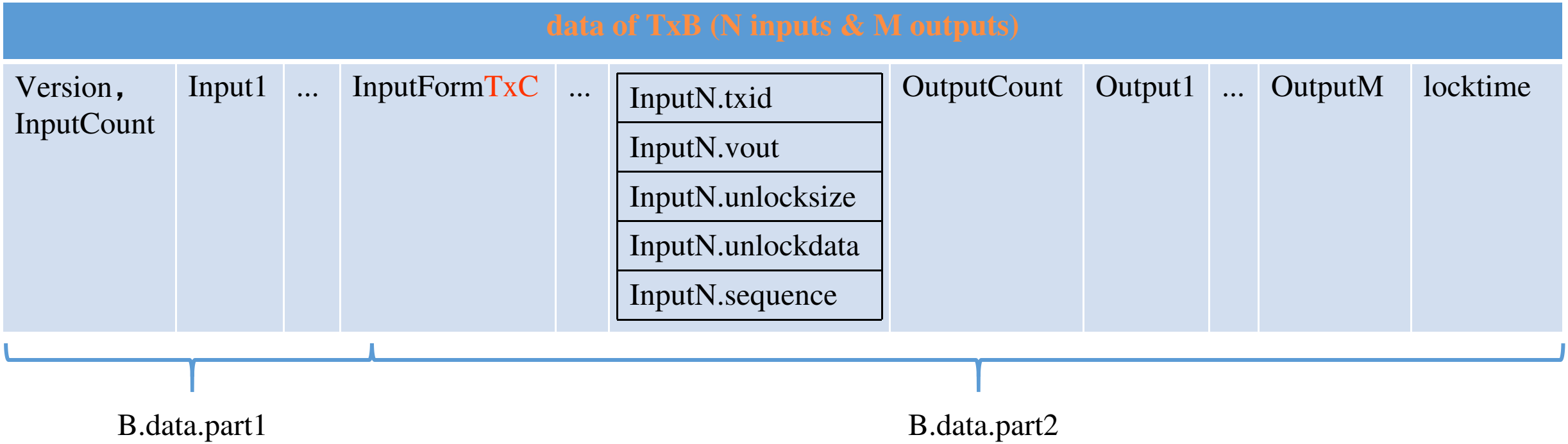
解决要点：

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，输入将分成两个部分，用红色框来表示：



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”