BOLT: A Bitcoin Transaction Latching Mechanism & Token Protocol

Preprint · February 2024

2 authors, including:

Frederick Liam Simon Honohan
Trinity College Dublin
1 PUBLICATION **0** CITATIONS

SEE PROFILE

BOLT: A Bitcoin Transaction Latching Mechanism & Token Protocol

Frederick Liam Simon Honohan
B.A, B.A.I, M.Sc, OSCP

February 11, 2024

Abstract

An electronic logic gate is composed of one or more inputs which combine to produce a single true or false output. Similarly a Bitcoin transaction can be composed of multiple inputs however they can create multiple outputs. At runtime, when processed, the computational programs contained in a transaction also, like logic gates, each produce a single binary result but the transaction can only be mined into a block on the chain if all of those programs return true like an *AND*-gate. This paper introduces a novel method to allow for a transaction to process information from not only its given inputs, but also the inputs & outputs of additional transactions. Also, a complete unbounded 'Simple Payment Verification' compatible token protocol which overcomes the 'Back-To-Genesis' problem is outlined as another use-case example of the new primitive.

1 Introduction

When processing transactions destined for the blockchain containing the records of who owns & spent what, Bitcoin miners must come to a consensus of the blockchain's full ownership state (without conditionality). Once a transaction has been submitted each of its inputs contain an outpoint reference to an unspent-transaction-output (UTXO) along with an unlocking attempt as a solution to the referenced output's locking conditions. The interfaces, which are the transaction inputs & outputs, each contain a piece of a computational program known as the 'unlocking script' or *scriptSig* & 'locking script' or *scriptPubKey/pubKeyScript* respectively.

During processing, the miners look up the unspent output & then prepend its locking script with the proposed input solution. The combination of the scriptSig followed by the scriptPubKey is then processed as one computer program which outputs true or false. Figure 1 is included hereunder to help the reader visualise the description of this process. In order for a transaction to be confirmed as true/valid, its referenced outputs to be marked as spent and the transaction's

A Clock Tick is All You Need [1]

SneakyFox

December 11, 2024

Abstract

An electronic logic gate is composed of one or more inputs which combine to produce a single true or false output. Similarly a Bitcoin transaction can be composed of multiple inputs however they can create multiple outputs. At runtime, when processed, the computational programs contained in a transaction also, like logic gates, each produce a single binary result but the transaction can only be confirmed into a block on the chain if all of those programs return true, like an *AND*-gate. This paper introduces a novel method to allow for a transaction to process information from not only its given inputs, but also, the inputs & outputs of additional transactions. Also, a complete unbounded 'Simple Payment Verification' (SPV) compatible token protocol which overcomes the 'Back-To-Genesis' problem [2] is outlined as another use-case example of the new primitive.

1 Introduction

When processing transactions destined for the blockchain, containing the records of who owns & spends what, Bitcoin miners must come to a consensus of the blockchain's full ownership state (without conditionality). Once a transaction has been submitted, each of its inputs contains an outpoint reference to an unspent-transaction-output (UTXO), along with an unlocking attempt as a solution to the referenced output's locking conditions. The interfaces, which are the transaction inputs & outputs, each contain a piece of a computational program known as the 'unlocking script' or *scriptSig* & 'locking script' or *scriptPubKey/pubKeyScript*.

During processing, the miners look up the unspent output & then prepend its locking script program with the proposed input solution program. The combination of the scriptSig followed by the scriptPubKey is then processed as one computer program which outputs true or false. Figure 1 is included hereunder to help the reader visualise the description of this process. In order for a transaction to be confirmed as true/valid, its referenced outputs to be marked as spent & the transaction's new unspent outputs to be recorded in a block on the chain, every one of its inputs' script & referred previous on-chain output script (output to input; new output or outputs) interface program pairs must evaluate successfully.

2 ScriptContext, sigHashFlags & OP_PUSH(C)TX

Bitcoin transactional programs are built from 'OpCodes', a set of executable logic operations defined by the protocols scripting/programming language 'Bitcoin Script'. The set of operations is akin to

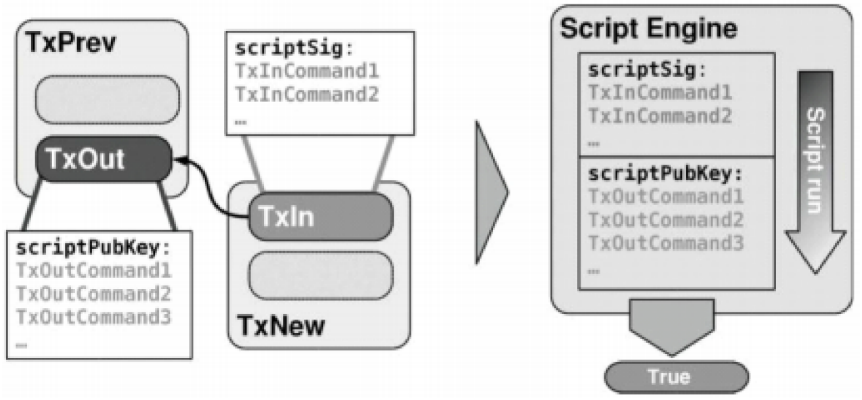

Figure 1: 'TxNew' referencing 'TxPrev' & the order in which the interface scripts are executed.

that of a CPU's instruction set into which programs are compiled & subsequently executed by the processor's array of logic gates. As an optimisation, the language includes specific ECDSA functions compressed into single opcodes to allow for low cost public key cryptography & reduce transaction sizes. It is typically by this standard cryptographic mechanism that most blockchain transfers are authorised. The most commonly used opcode of the cryptographic variety is *OP_CHECKSIG*.

To unlock a standard pay-to-public-key UTXO in order to reallocate some amount of Bitcoin locked into Alice's possession (her public-key) she must notify the network of her intent by crafting & broadcasting a transaction. Said transaction must contain an input referencing her aforementioned unspent output along with a scriptSig containing an authorising signature from her private-key. The data she needs to sign in order for the signature to be successfully validated by *OP_CHECKSIG* is a 32-byte hash value calculated from a transactional context blob of data, first known as the *SigHashPreimage*, now more accessibly known as the *ScriptContext* [3]. The *PUSHTX* technique ought to be renamed PUSHCTX for the same reason, as a transaction can contain multiple inputs each with different contexts.

ECDSA signatures, to be validated by the set of *CheckSig* opcodes (namely, *CHECKSIG*, *CHECKSIGVERIFY*, *CHECKMULTISIG*, & *CHECKMULTISIGVERIFY*), necessarily need to be made over the double SHA-256 hash value of the ScriptContext. Depending on a user defined flag (denoted as sigHashFlag), the ScriptContext data blob can contain several different permutations of various pieces of information about the currently executing transaction program. The ScriptContext includes the currently executing output's outpoint & locking script (in whole) itself (scriptCode) as well as other valuable information summarising the executing transaction's context as in Figure 2. It is noted in [4] that 'different inputs can use different SIGHASH flags enabling complex compositions of spending conditions.'

2.1 The *PUSHTX* technique

After the release of nChain's White Paper #1605 *'PUSHTX & its Building Blocks'* [5] executing transaction interface scripts could reliably inspect every piece of data afforded to them by a given ScriptContext serialised data structure. Given ScriptContext data structures can be validated as legitimate by way of a nifty ECDSA OP_CHECKSIG reflection-like known-private-key trick, a

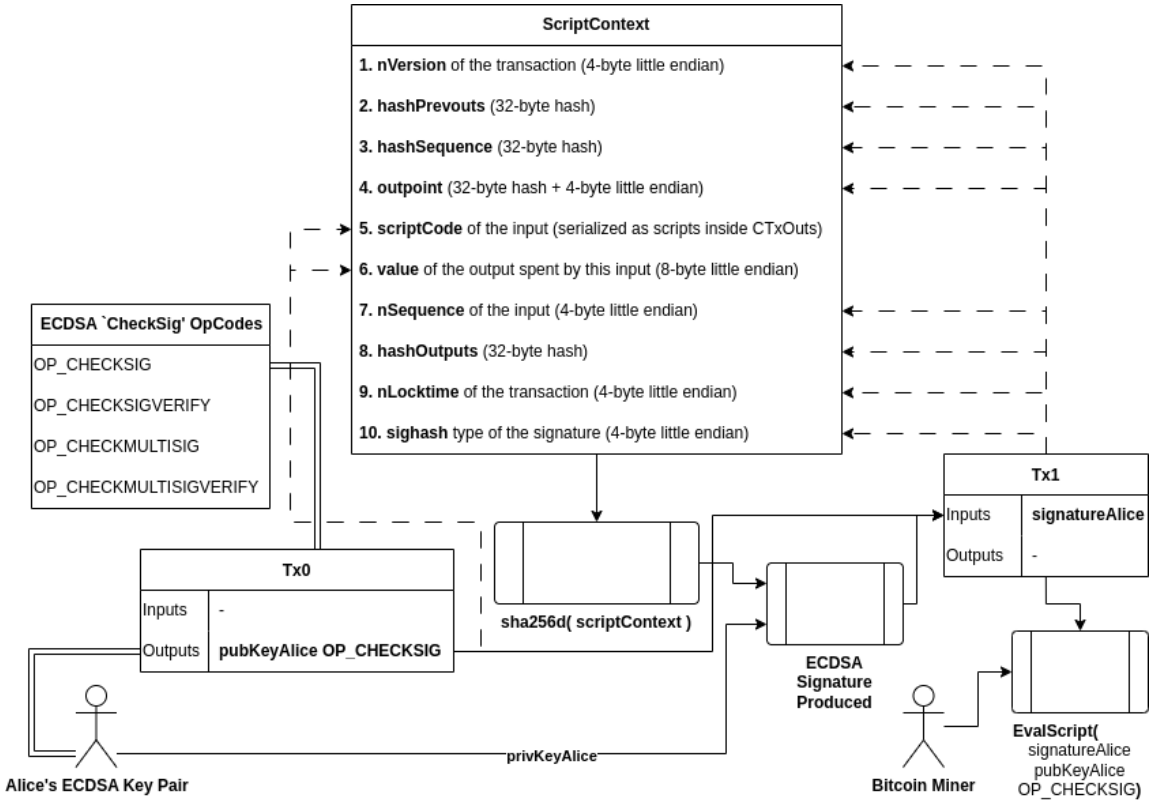

Figure 2: The ScriptContext data structure & ECDSA Signature process governing its resultant hash value.

description of which is beyond the scope of this paper.

Before that whitepaper was released, knowledge of runtime access to that valuable information was unknown. Such transactional context was never thought of as practically accessible to the currently executing transaction interface scripts themselves. The PUSHTX technique affords the transactor the ability to enforce advanced unlocking requirements including for specific output outpoints to be spent/unlocked alongside one another, or even for the transaction's newly created outputs to conform to be precise specific types of locking script.

Since signatures are put in an input's scriptSig, a ScriptContext blob cannot contain information about the scriptSig into which the signature over it is required to be placed into the transaction. Nor can the ScriptContext contain any reference to any of the other transaction's input scriptSigs either. The drawback of this (unfortunate) impossibility is two fold. Any previous locking script & current unlocking script combination cannot access verifiable information about any of its transactions neighbouring input scripts. Furthermore, the history of a chain of UTXOs which is directly accessible by way of the PUSHTX technique at processing time/runtime is limited to one transaction only.

To overcome these drawbacks, another, new, building block is required in order to develop a token protocol for which history can be proven via mathematical induction.

3 The Latching Mechanism

Since Bitcoin was designed with the predicate true/false validation system, it makes sense to look at transactions as a whole & also their inner script programs themselves as logic gates. Each transaction has one or more inputs & one or more outputs. Further, each input can contain an varying number of data variables to try satisfy the referenced output locking conditions in order to produce a binary result. Finally, the UTXOs referenced in these inputs are proof that the transaction in which they were created contained successfully unlocked inputs.

In order for an executing transaction script to inspect other scriptSigs, a novel technique to connect interface programs has been developed. By utilising values obtained from the PUSHTX method, an encapsulating set of features define the method.

A 'Bolt' is any UTXO with involves the following three requirements:

1. The successful unlocking of at least one additional UTXO depends on the simultaneous unlocking of it, the *boltTxo*, &/or visa-versa*.

2. The full raw serialised transaction which created the *boltTxo*, the *boltTx*, is used to calculate the *boltTxid* during the concurrent processing of the set of dependent UTXOs.†

3. The *boltTxid* is then used to prove the *boltTxo* & the other UTXO(s) are executing concurrently in the same transaction.†

*One or more references between the entire UTXO set are required, made by directly referencing known UTXO outpoints, if possible, derived from common ancestry, introduced at runtime as input or by any other manner.

†It seems natural that requirements 2 & 3 need to be calculated consecutively in the same program, but it can be done by all UTXO spending programs in a sort of multi-bolt-lock & other proofs by inference will exist.

Another way of looking at the system is to consider at least two transactions. A *latchTx* when the requirement of the unlocking of the *boltTxo* is applied to the other UTXO. Following that, an *unlatchTx* is made when the boltTxo is spent & enabling the unlocking of the other UTXO. Finally, demonstrating the versatility of the order of the transactions in the system, a boltTxo might have been created before, during or after the latchTx (which is naturally always before the unlatchTx, obviously).

Due to the nature of this multi-transaction, multi-program linking, the validity of multiple executing scripts can be made to depend on all or some combination of each other in a distributed system. If the unlatchTx is accepted by the network as valid it can be safely inferred that both (or all neighbouring) input scriptSigs satisfied their previous outputs locking conditions.

By implementing these unique requirements upon a set of UTXOs further useful contextual information is afforded to the executing transaction. Importantly, the previous outpoint reference & scriptSig of every input of the boltTx are exposed to inspect & control program flow if required. Furthermore, all of the output pubKeyScript & values including the boltTxos can be inspected. In fact if any of the scriptSigs in the boltTx's inputs include ScriptContext blobs satisfying a PUSHTX requirement then access to inspect & operate on further ancestral pubKeyScripts is also possible. This significantly improves on the single pubKeyScript made available with the PUSHTX script context technique.

A bolt can be thought of simply as a UTXO left over in one transaction so that another later transaction can inspect the transaction containing it.

4 Configurations

In order to simplify things, the following examples will use one Bolt linked to one other UTXO (which we will identify as the 'Token'), in various different ways. Also for ease of comprehension, in all of the examples, except where distinctly noted, the dependent Token enforces all of the method's requirements whereas the Bolt acts merely as a consumable pointer upon which the Token's conditionality rests. The division of the responsibility to enforce the method's three distinct requirements between the Bolt & the system's other UTXO(s) is not strict. Several use case examples will now be described in order to illustrate some of the method's different modes of referencing, internal operations & also various transaction & UTXO orderings.

4.1 Disconnected Services

A Token can only depend directly, by outpoint reference, on a Bolt that already exists, otherwise the referential link must be made post-hoc. Both of these time-based configurations are illustrated in the two different disconnected bolt configurations outlined in Figure 3.

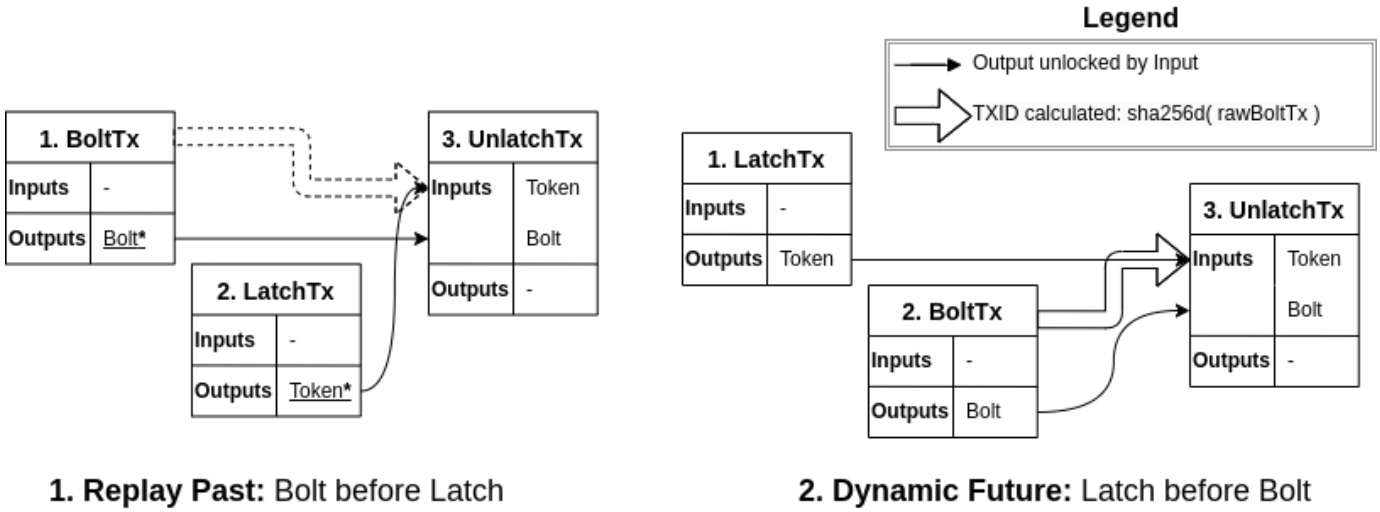

Figure 3: Different time based configurations of disconnected Bolts ("MicroServices").

4.1.1 Replay Fixed/Known/Past BoltTx

On the left a bolt has been left over in the appropriately labelled BoltTx. Since it existed before the latch mechanism was applied, the bolt outpoint can be hardcoded into the latchTx. This is highlighted by the underlined & asterisked Token & Bolt outputs in Tx 1 & 2 in Figure 3.

This demonstrates that one can apply a lock on a UTXO to be spent alongside another by simply using the UTXO's outpoint in the PUSHTX hashPrevouts set check. Therefore, applying the bolt method here is merely useful for transaction inspection but is overkill for a simple paired UTXO lock. It is, however, a very important piece of the protocol puzzle, affording novel further configurations & a detailed use-case & extension will be described later.

4.1.2 Dynamic Future BoltTx

Also illustrated in Figure 3 is a second configuration which is more powerful because it allows for dynamic outpoint dependencies. This is the perfect configuration for Oracles, or off-chain data reliance. A simple system whereby a Token issuer can make simple optional protocol upgrades at any time in a Token's future is outlined now as another use-case example.

Given a self replicating token protocol which contains identifying features such as a *genesisOutpoint* & an *issuerPubKey*, for example, a solution to enable safe opt-in protocol upgrades can be built.

The Token issuer, such as a video game company, for example, can announce via their blockchain connected micro-transaction games that their sub-protocol's tokens are due an optional upgrade. When a user decides to accept the upgrade, a completely disconnected boltTx can be made by the issuer containing a set of different data items for inspection by an upgrading token program in the following way:

1. An *upgradePubKeyScriptTemplate* serialised data structure containing the information needed to swap out the old program in favour of the new one.
2. A reference to the genesisOutpoint & issuerPubKey contained in the Token.
3. An issuerPubKey verifiable signature over the payload contained somewhere in the boltTx.

The user can then action the consumption of the bolt in a protocol upgrade transaction. The Token would make use of the items from the boltTx when resolving the bolt, perhaps in the reverse manner, as follows:

1. Ensure that the transaction contained a valid signature from the issuerPubKey before continuing.
2. Then the genesisOutpoint & issuerPubKey contained in the boltTx is confirmed to match that of the Token.
3. The upgradePubKeyScriptTemplate serialised data structure would then be parsed & utilised to create the new Token pubKeyScript by way of the hashOutputs PUSHTX technique.

This system is only described above in broad terms. Other standard signature checking & transaction operations would of course be involved & the upgrade might involve more than one pubKeyScript (e.g. by way of a chain of upgrade transactions).

4.2 Connected Services

A third configuration, a slight extension of the method with the future in mind, is described underneath in Figure 4. It shows a simple extended setup one can attach to the bolt mechanism in order to transfer control of a UTXO without actually needing to unlock it. That is to say, by attaching the latch mechanism to a boltTxo to be created in the future by a related descendant of the system, Alice may be able to transfer control of the Token to Bob.

Bitcoin does not naturally have an inbuilt request response type handshaking model built in. This example outlines one whereby some Tokenised assets, or even just Bitcoin, can be transferred, but only after the recipient completes an acceptance action. It works as follows:

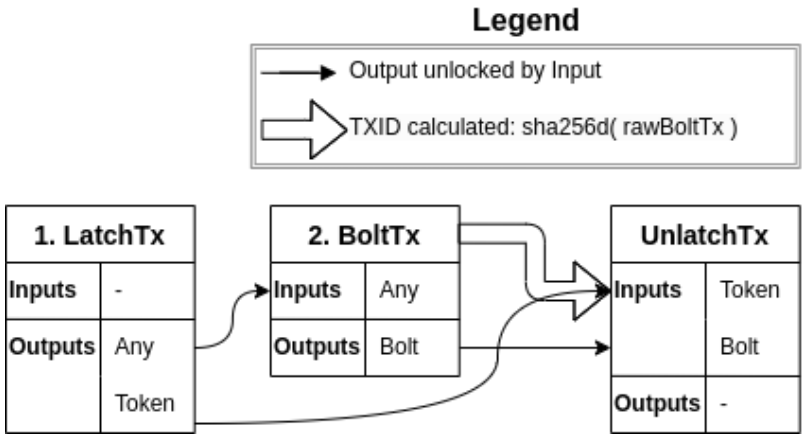

Figure 4: Control transfer by related descendant e.g. Bolt from Future!

1. Alice creates the LatchTx with the Token output alongside another output marked in Figure 4 as *Any*'.
2. In order to unlock the Token it requires to spend a Bolt from & thus inspect the BoltTx which must include the Token's neighbouring UTXO, Any, as an input.
3. Alice can then make the BoltTx, designating the newly created Bolt into Bob's control.
4. In order to collect the Token left over in the LatchTx Bob can unlock the Bolt from the BoltTx & complete the UnlatchTx & take control of the Token in question.

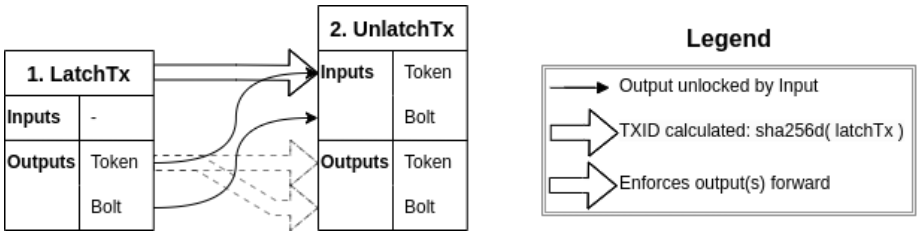

Figure 5: Recovery latch mechanism, with the Token enforcing both the Bolt &the Token itself forward.

4.2.1 Recoverable Access Tokens

Imagine a Bitcoin based simple access token service for which authorised access can be granted to the current owner of a token by providing a signature from the key it is locked to, over some select data. Such a token would be a virtually free blockchain based mechanism for automated inter service authentication as no transactions need to occur. But where key rotation is a common security measure however, transfer-ability of the token is a must. Combining the hot & cold

wallet concepts with the bolt method & techniques afforded by PUSHTX, a novel mechanism to permanently secure a recovery mechanism of the funds can be built. One might describe this setup as akin to parallel-processing, as appears in Figure 5.

A hot wallet is the term used to describe a signature service with low security e.g. high availability on an internet connected server (always susceptible to security breaches via social engineering, penetration or otherwise). A cold wallet is the opposite in security terms, the key is hidden securely offline & retrieved only when necessary, e.g. from a vault buried a kilometer underground! The Bolt in this configuration can be visualised as a sort of lazy latching hook. The system which enables permanent recovery of itself to it's rightful owner is setup as follows:

1. The Token is locked into possession of the hot wallet key & the first requirement to unlock the Token is the production of a signature from that key.

2. When transferred via valid signature, the Bolt required by the Token must be present in the same transaction & simultaneously unlocked. This is the Token's second requirement & is validated using the outpoint & hashPrevouts values granted to the system the PUSHTX method.

3. In this normal transfer mode, the Bolt merely needs to be certain of the polar opposite, namely, that the Token is present & being unlocked in the same transaction (no signatures are needed).

4. One or both of the Token & Bolt pair can enforce their distributed program forward into the next transaction's outputs, validated using the hashOutputs PUSHTX value.

5. Upon a security breach, the requirements for each UTXO to be recovered, in this example, are effectively the reverse of the normal mode of operation.

6. The Token, now locked to the hacker $UPERH4CKERMONST@DEW420's possession, can be recovered via backdoor, utilising this bolt method.

7. A signature hardcoded cold wallet derived publicKey in the Bolt can be used to unlock it & return control of the Token back to the autonomous system.

8. The sysadmin arduously digs up the vault & produces a signature to unlock the Bolt with the cold wallet key & provides new information to reassign control of the system.

9. The Token is simultaneously unlocked in the same transaction, this time without a signature but merely by validating that the Bolt is being unlocked alongside it via the provided PUSHTX context, & it will understand by inference that it is being recovered.

10. $UPERH4CKERMONST@DEW420 is sent to prison due to the permanent blockchain record of the criminal offense.

Interestingly, the mechanism described above could be enforced with a single UTXO & thus single blockchain program. Visualising the condensed construction of a single self replicating UTXO program serves to demonstrate how the Bolt method can be used to distribute the computation of it's requirements, or indeed the requirements of any other program. Multiple, linked, multi-directionally or otherwise, co-dependent UTXOs can interoperate to achieve such a distribution.

Despite that, the use of the multiple UTXO Bolt method in this way, is not in fact just overkill, but may also be ill-advised. The defining feature of the Bolt method's construction is that the full raw transaction that created the Bolt is needed by the Token in order to calculate the boltTxid. Since the Token & Bolt are to be spent in the same transaction, by necessarily requiring the inclusion of the full raw transaction, the size of the system's transactions begins to bloat as shown in Figure 6.

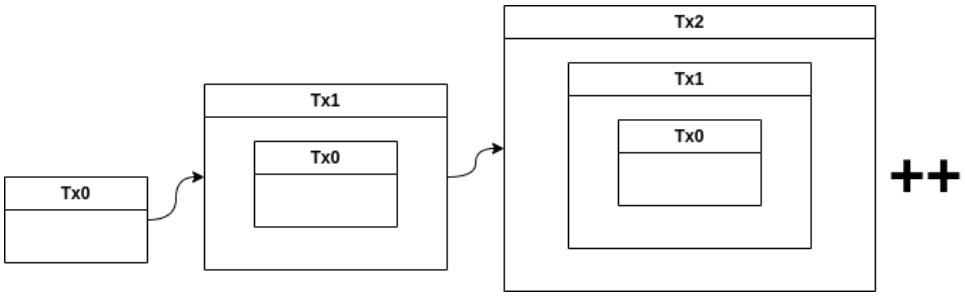

Figure 6: Transaction history bloat problem

The Token must be provided with the transaction that created the Bolt i.e. the transaction that created itself. Ergo, after ten transfers the transaction containing the token UTXO will contain all ten of the previous transactions, which leads to the continuous growth of transaction sizes. This bloat issue lead to the Bitcoin development community defining & naming the bloating problem as the *Back to Genesis Problem.*

5 Some History

Once the byzantine general's problem was solved, creating a self administering network of ledger keepers economically with the Bitcoin protocol, uses for the technology effectively pigeonholed into token/digital currency issuance. The most famous asset to be tokenised to date is a utility U.S. Dollar "pegged" issuance named Tether (USDT). Arguably it's existence & historical issuance activity indicates that it is merely the fractional reserve banking establishment gaming the system, or by shysters simply printing more money.

Tether initially was issued via a Token protocol atop Bitcoin named OMNI-Layer, a very basic token protocol for which specific wallets were used with the issuers keeping track of the funds off-chain. This is the main drawback with other Bitcoin based token protocols in widespread use today; The legitimacy of the tokens relies solely on the issuer's secondary index & the Bitcoin miners themselves merely keep track only of the blockchain transactions & cannot actually attest to the validity of the token's contents. The protocols rule-sets simply piggyback Bitcoin's mechanism of ownership transfer & little else. As noted in [6]: *"the blockchain is used just to notarize the actions that manipulate tokens, but not to check that these actions are actually permitted."*

Or, in other words, the legitimacy of the funds claimed by any token resides completely off chain & this is still true even for the Ordinal token protocol on BTC today in 2024. Initial frustration surrounding this seeming limitation of Bitcoin (amongst other things like curiosity & greed) led to a number of competing public blockchain protocols like Ethereum (ETH) & Cardano (ADA) & all the others.

However, Bitcoin's inventor had already stated: *'I wanted to design it to support every possible transaction type I could think of... The solution was script, which generalizes the problem so transacting parties can describe their transaction as a predicate that the node network evaluates... It's just an equation that evaluates to true or false.' [7]*

Therefore since a computer program can simulate a computer processing unit (CPU) processing an instance of another program (or itself), this can also be done (in an immutable timestamped manner) with Bitcoin. Its transactional programs can be combined, much like transistors in a CPU (as mentioned in the abstract) & visualisation of this ought to illustrate the Turing completed-ness of the Bitcoin protocol itself. Simulating anything with any system is the test used to determine whether that system is Turing complete.

Despite this, Ethereum introduced account based (bottleneck) "self-executing" contracts & token specifications. Then one of ETH's founders, having realised the mistake (bottleneck) reverted to the original Bitcoin UTXO system with Cardano (ADA) & introduced some native token rules to it's protocol via a second consensus layer. For reference, according to ChatGPT today the total combined market capitalization of BTC, ETH, ADA, & USDT: '...is approximately $1.848 Trillion.' [8].

In what might go down in history as one of the grossest acts of engineering perversions affecting the Universe to date, malicious actors/legacy-establishment sought to choke the chain & commandeer the Bitcoin protocol, attacking it via a soft fork mechanism (a kind of gameable democratic voting procedure) introducing a "new-feature" in 2016, something nicknamed 'SegWit' (short for segregated witness). The feature broke two well-established engineering maxims:

1. If it's not broken, don't fix it.
2. Keep it simple, stupid (the *KISS* principle).

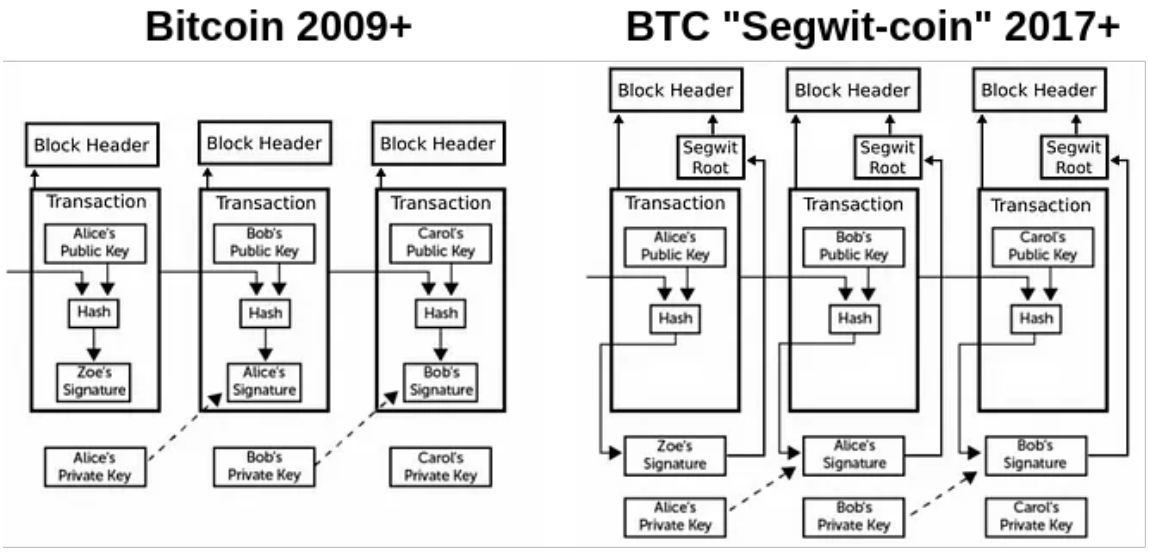

Figure 7: Block & transaction structure differences introduced by SegWit.

The simplicity of both the original block structure & transaction format is illustrated in Figure 7. Not only did SegWit make things evidently more complicated in the Bitcoin protocol, it required the inventor's definition of their own invention to be undermined! The first sentence of the second section of the Bitcoin whitepaper defines a coin as: *'a chain of digital signatures'* but as can be seen in Figure 7, the signatures (scriptSigs) have been taken out of the SegWit transactions entirely. Thus, since they no longer reside inside the transaction they are no longer a part of the calculated *txid* & since that txid is one aspect of the information that is required to be signed to perform a transaction the chain of digital signatures is severed.

Surely, one would think, the inventor, perhaps, if alive, would have stepped in to stop such a grievous affront to their invention? More on this later.

Nearly all USDT is issued on chains other than BTC today because it's utility value is, as the CEO of J.P. Morgan bank Jamie Dimon put it, that of a digital *'pet-rock'*. Mr. Dimon might be pleased (or dismayed) to know, however, that counterfeit-proof, "self-executing", inter-operable & upgrade-able tokens (as it turns out) were possible right from the moment of creation of the inaugural Bitcoin genesis block back in 2009.

6 The *Back-To-Genesis Problem*

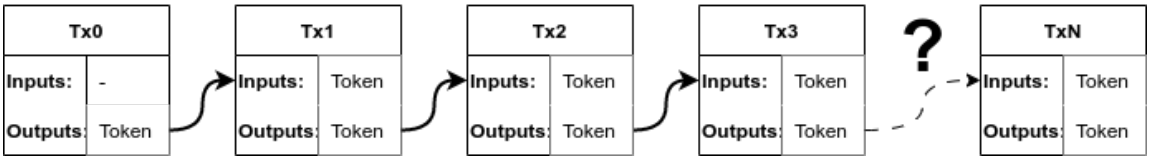

Figure 8: The *Back-To-Genesis Problem*

Once some USDT is minted on the blockchain its legitimacy depends entirely on the issuer's signature action &/or also the data contained in the genesis transaction. Whoever issues the cash-like currency tokens denotes those quantities & proclaims those quantities to be redeemable. As a token moves forward in time it creates a growing history of transactions from its genesis & in order to fully validate the legitimacy of any purported tip of the token's history & that history is required in totality. Another way of looking at it is; Given an unspent transaction output, is it linked in an unbroken chain of unlocked inputs & locked outputs through some count of transactions to the genesis outpoint & issuing governance it purports to be from?

Ethereum does not have this problem because it has a reliable memory pointer represented as the account balance & its miners must come to a consensus of any changes to everyone's balance every block & this extra work affords that protocol the claim to self-execution. As mentioned previously Cardano uses a second layer to achieve such consensus but some lateral thinking is necessary to achieve the same on the first layer of a UTXO based blockchain without bloating transaction sizes.

The problem is illustrated in Figure 8. Without introducing a novel solution (for example, introducting a proof by mathematical induction) the only way for the Bitcoin network or any other entity to know that the token in the output at TxN came from the issuers mint action at Tx0 is to have knowledge of the full transaction history while processing or validating its legitimacy. An alternative reconfiguration of the bolt primitive combined with some other rules can provide us with an induction proof based token protocol: Utilising multiple inputs & outputs, & by way

of induction, can prove a Tokens legitimate identity with the existence of two unspent transaction outputs in two separate distinct transactions.

That is to say, with merkle proofs of two related UTXOs, their ancestry from their recorded genesis outpoint is inductively proven at the transaction processing layer no matter how many transactions the token has moved since genesis.

7 The B.O.L.T Protocol

The ***'Bitcoin-Original-Layer#1/Lightning-Token' protocol*** contains another reconfiguration of the building block surrounding the operation & relationship between the linked transaction outputs which demonstrates the versatility of the mechanism.

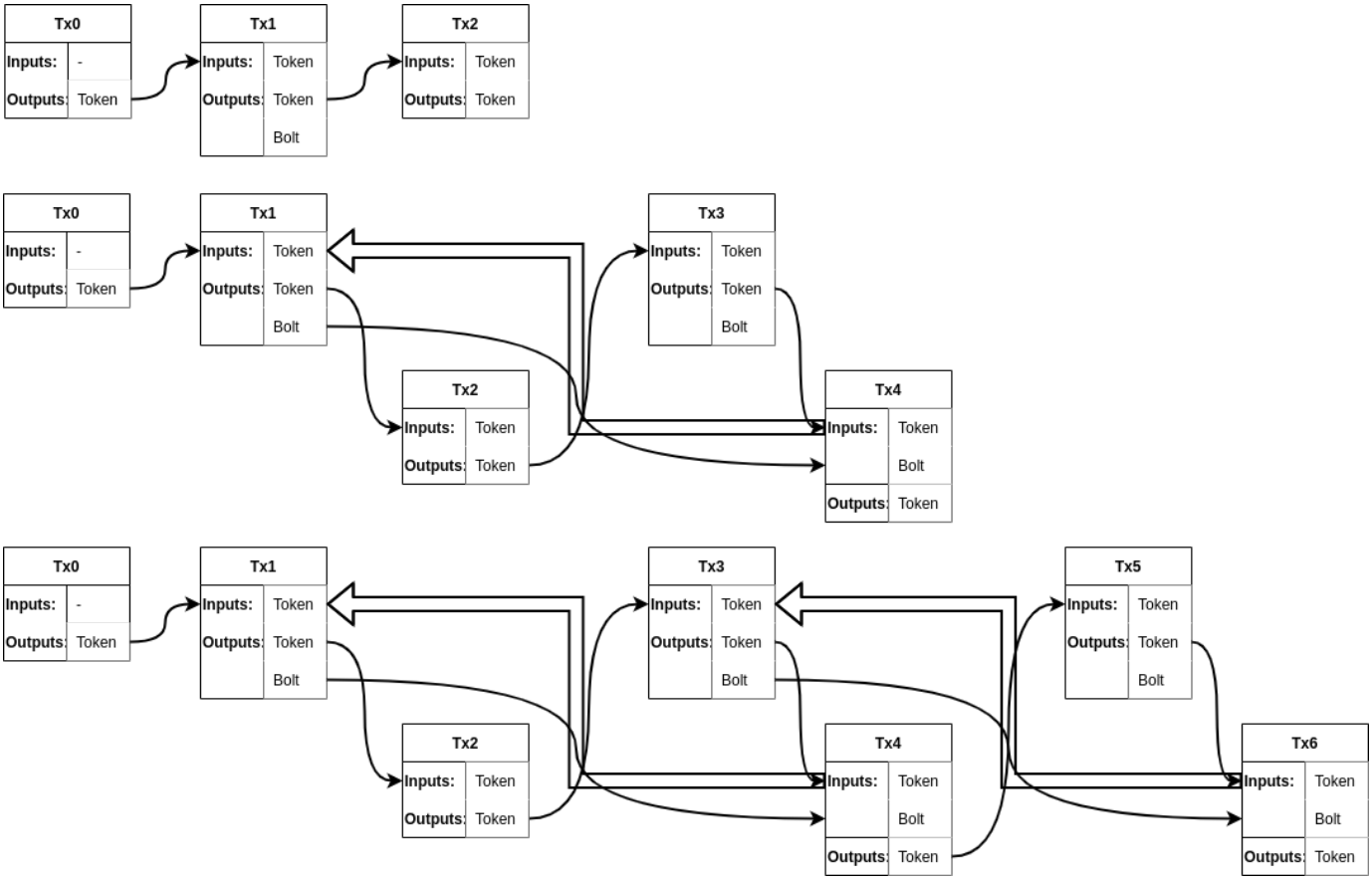

Figure 9: A BOLT-based back-to-genesis token protocol solution

7.1 Authenticity Proof by Mathematical Induction

Define $P(N)$ as the property that for a transaction Tx_N where $N \geq 3$, a token maintains an unbroken chain of transfers back to its genesisOutpoint &/or came from a specific issuerPubKey, under the given conditions. N begins at 3, however it is trivial to prove the cases for $N \leq 3$ given that the genesisOutpoint is equal to the grandparentOutpoint at Tx_2 & the parentOutpoint at Tx_1.

We define the property $P(n)$ as follows:

$$P(n) : \text{At the } n^{th} \text{ transaction}, \begin{cases} \text{genesisOutpoint} \in \text{Tx0}, \\ \text{grandparentOutpoint} \in \text{Tx}(n-2), \\ \text{parentOutpoint} \in \text{Tx}(n-1), \\ \text{hasBolt} \in \{\text{TRUE}, \text{FALSE}\} \end{cases}$$

Base Case 1: Odd Transaction (Tx3)

Consider $P(3)$ with Tx_3 as the base odd case.

$$P(3) : \text{At Tx3} \begin{cases} \text{genesisOutpoint} \in \text{Tx0}, \\ \text{grandparentOutpoint} \in \text{Tx1}, \\ \text{parentOutpoint} \in \text{Tx2}, \\ \text{hasBolt} = \text{TRUE} \end{cases}$$

In this case, hasBolt = TRUE for Tx_3. Since its grandparent Tx_1 contains a token spend directly from Tx_0 & since spending the bolt from Tx_1 & rebuilding that full transaction containing the spend from the token's greatGrandparent/genesisOutpoint, in an input (& the grandparent token output) in memory, it proves an unbroken chain back to the stored genesisOutpoint. Which has been enforce-ably passed down by requirement.

Base Case 2: Even Transaction (Tx4)

Consider $P(4)$ with Tx_4 as the base even case.

$$P(4) : \text{At Tx4} \begin{cases} \text{genesisOutpoint} \in \text{Tx0}, \\ \text{grandparentOutpoint} \in \text{Tx2}, \\ \text{parentOutpoint} \in \text{Tx3}, \\ \text{hasBolt} = \text{FALSE} \end{cases}$$

In this case, hasBolt = FALSE for Tx_4. Since its parent Tx_3 contains a token spend from Tx_2 & a bolt, the SigHashPreimage given in the Tx_3 input locking script code from the token in Tx_2 contains the genesisOutpoint as the grandparentOutpoint. This requirement necessitates confirmation by the imposed future restriction on the token at Tx_5 & as the protocol requires two transactions to transfer control, it is impossible to counterfeit a token & a bolt to oneself, therefore a token counterfeit attempt can never be transferred.

Inductive Step

Assume $P(k)$ is true for some $k \geq 3$, i.e., Tx_k, & assuming the presence of a token in Tx_k, Tx_k maintains an unbroken chain back to the genesisOutpoint. We need to prove $P(k+1)$ for Tx_{k+1}.

$$P(k+1) : \text{At Tx(k+1)} \begin{cases} \text{genesisOutpoint} \in \text{Tx0}, \\ \text{grandparentOutpoint} \in \text{Tx}_{k-1}, \\ \text{parentOutpoint} \in \text{Tx}_k, \\ \text{hasBolt} \in \{\text{TRUE}, \text{FALSE}\} \text{ for } Tx_{k+1}. \end{cases}$$

Consider two cases based on the value of 'hasBolt' for Tx_{k+1}:

Case 1: Tx_{k+1} has a bolt (hasBolt = TRUE).

If Tx_{k+1} is an odd-numbered token transaction it would also contain a bolt output, its grandparent Tx_{k-1} contained a token, & also must have a bolt. Using the inductive process, & assuming the presence of a token in Tx_{k-1}, Tx_{k-1} maintains an unbroken chain back to the genesisOutpoint & this is proven at layer-1 by the token's protocol. Therefore, Tx_{k+1} also maintains this unbroken chain.

Case 2: Tx_{k+1} does not have a bolt (hasBolt = FALSE).

If Tx_{k+1} is an even-numbered transaction, its parent Tx_k maintains an unbroken chain back to the genesisOutpoint, assuming the presence of a token & a bolt in Tx_k by way of the bolt restriction applying in Tx_{k+2}. Since Tx_{k+1} directly follows Tx_k &, crucially, there exists a token & a bolt in Tx_k, the token at Tx_{k+1} preserves the unbroken chain.

7.2 Proof Conclusion

$P(N)$ is true for all transactions Tx_N where $N \geq 3$ & since Tx_3 has a direct link to Tx_0 by way of outpoint input to (Tx_1), Tx_N where $N \leq 3$, a token maintains an unbroken chain back to the genesisOutpoint. In practice, a token can be validated by ensuring that a bolt has been left behind in one of its claimed ancestral transactions & that a token existed in that transaction also. Those actual facts are validated by the miners at runtime. Remember also that in order to counterfeit a token, one would need to counterfeit an ancestor which is an impossibility because that token would also need a counterfeited ancestor etc, etc. Descendants cannot possibly exist without their own ancestors. Or, as the philosophers might note; 'It's turtles all the way down!".

7.3 Fungibility

The base BOLT protocol can be extended to incorporate fungible characteristics & these are outlined in the following graphs along with appended demonstration contract code written in Elas technology's custom proprietary compiled Bitcoin programming language "Script Extended" (.sx) in the Appendix section.

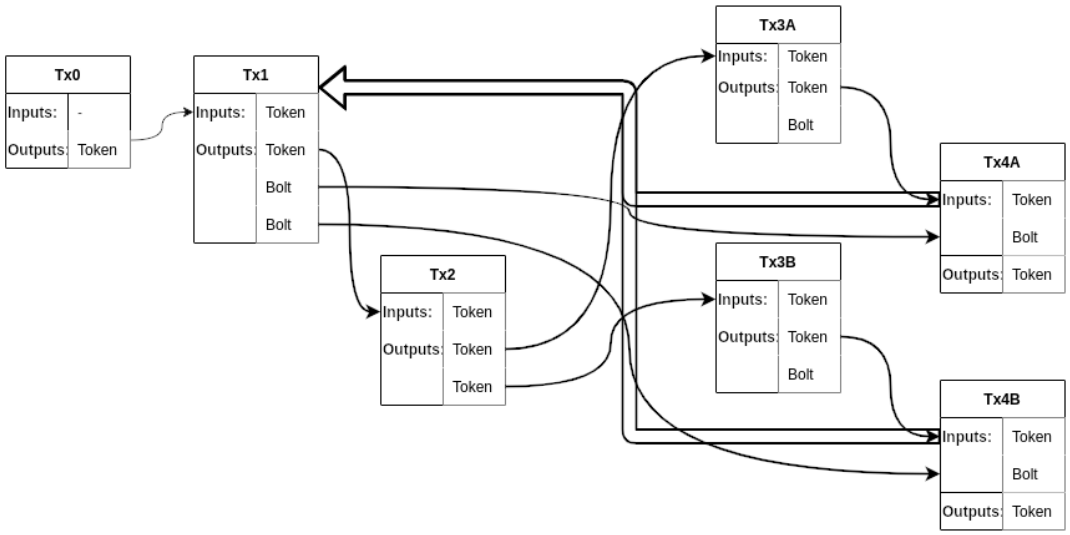

Figure 10: BOLT Protocol Fungible Split Example

Tokens can be minted with a balance & that balance can be divided between two recipients per actioned event like Ethereum tokens but on a UTXO blockchain which can scale to accomodate all cryptocurrency transactions & will not succumb to the congestion problem experienced in account based blockchain designs (see the "CryptoKitties" fiasco [9] & cost of transaction today).

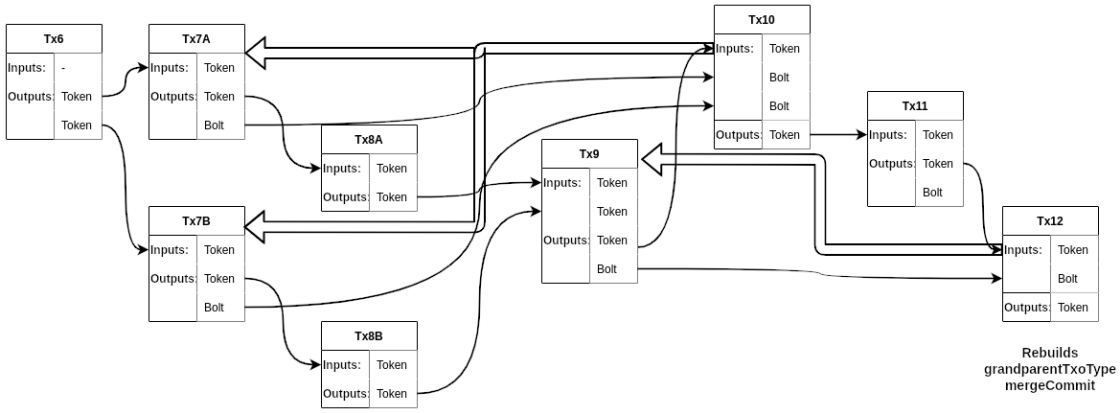

Figure 11: BOLT Protocol Fungible Merge Example

Also, the recipients can merge their balances in another event, in which a new subsequent token is created with a balance equal to the sum of the balances of the two recipients' tokens.

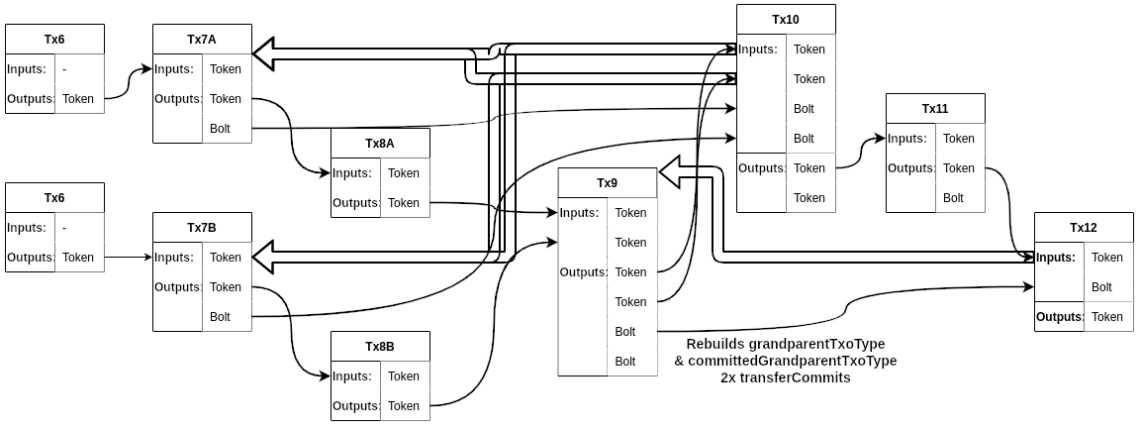

Figure 12: BOLT Protocol Fungible Swap Example

Finally, tokens can be traded, *aux bureaux de change*, between recipients of tokens created by two seperate issuing entities for an agreed exchange rate. Importantly it should be highlighted that the fungible & non-fungible code examples are included only to illustrate the processes of both type of BOLT protocol & are not yet fully tested for production release.

8 Conclusion

Debate rages on about which Bitcoin is the real Bitcoin, however these novel Layer-1 tokens serve to confirm which is real. Ironically, another debate is ongoing today about whether or not to re-enable a set of opcodes on the BTC network which were disabled by Satoshi in order to temporarily disable memory bomb disruption of service attacks [10]. In particular, the BTC developers are talking about the re-enabling of OP_CAT (the PUSHTX operation is impossible without that specific opcode).

Characteristically of the inventor (who originally disabled that set of opcodes), he re-enabled them when forking away from (the SegWit destroyed chain that is) BTC. Bitcoin became known as BCH for a while, until, again, Bitcoin sustained a takeover by manipulation as the BCH developers decided to re-modify the protocol. Foolishly they decided to add a processing validation bottleneck known as Canonical Transaction Ordering or CTOR. This forces the miners to reorder transactions in a block rather than keep the valuable order information (imagine unlatchTxs before latchTxs, how amusing). Bitcoin again forked away from such nonsense & is now traded under the ticker BSV, standing for Bitcoin Satoshi's Vision.

Bitcoin	Birth Year	Signature Chain	OP_CAT+	Tx Order(No CTOR)
Original (BSV)	2009	✓	✓	✓
Core (BTC)	2017	×	×	✓
Cash (BCH)	2018	✓	✓	×

Table 1: Comparison of Bitcoin Feature Sets

The coordinated SegWit & CTOR attacks on the blockchain & its novel time-machine-like

properties can easily be viewed as an act of war against the public by the legacy establishment to stamp out a financial revolution. As the inventor of the blockchain said recently *'Bitcoin was created for everyone'* [11]. Much like how mankind only needed one hyper-text transfer protocol (HTTP), it really only needs a single utilitarian blockchain protocol to provide a digital timestamped Great Library of Alexandria-esque source of information to compete with the private banking system.

BSV developers continue to build tools for a nice, peaceful wealth redistributing revolution, wealth which the central bankers are continuing to just print out of thin air.

This invention serves to objectively demonstrate that mankind needs only one blockchain implementation and that is the original protocol surviving today under the trading ticker BSV. It is therefore proven that all other blockchain implementations have Ponzi-scheme characteristics, in other words are scams. As Roger Ver pointed out only a day ago [12], the real Bitcoin protocol is the original Bitcoin; And given it's Turing-completedness, the BOLT token protocol can be extended to emulate privacy features akin to that of Monero & Zano cryptocurrencies & this protocol cannot be deployed or replicated on the competing Bitcoin blockchains of BTC, BCH etc, all of which are corrupted.

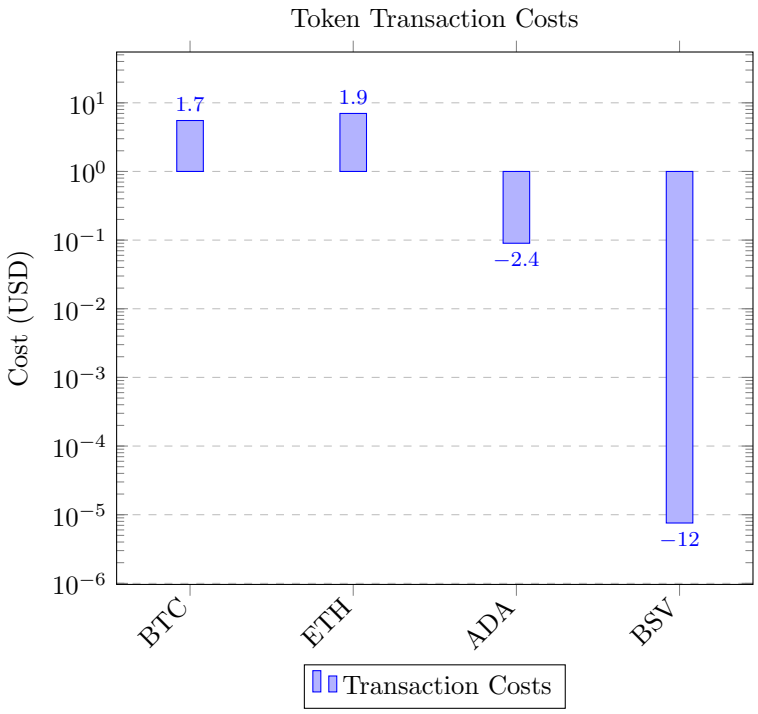

Figure 13: Token Transaction Costs Comparison (log scale). BTC: Bitcoin Ordinals, ETH: ERC20 tokens, ADA: Cardano tokens, BSV: BSV tokens

Currently Bitcoin-SV is the blockchain on which it is cheapest to transact & the only blockchain that can scale to realise the invention's full potential. It's tough to discern from the difference between the entries in the logarithmic graph in 13 but it's worth mentioning that it is nearly 12,000 times more expensive to make a token transfer on Cardano (developed by Charles Hoskinson [13],

[14], an Ethereum founding developer clamouring for control of this emerging technology), than the original Bitcoin (SV) developed by Australian Computer Scientist Craig S. Wright [15]! Furthermore, hilariously, the BOLT protocol could be emulated on Cardano with a single protocol change rendering it's entire *raison d'être* (a blockchain with layer-2 tokens) completely and utterly defunct/worthless which is really shocking, considering it's current market capitalisation of $37.75 Billion USD [16]!

9 Acknowledgments

I would like to extend my special thanks to those who have contributed significantly to the development & completion of this work. I am particularly grateful for the entire team at Elas PTY for their invaluable support throughout this project. Further, I would like to thank my family for their love & support throughout the years.

It is impossible not to conclude that after the implementation of SegWit (corrupting the chain of cryptographic signatures which define a coin as noted in [15]) coupled with artificial block-size limits; BTC ceased to follow the Bitcoin protocol, ruining it's ability to correctly function ever since block height 481,824, or **block hash**:

2ef9122d13e2d7803b6ebda48d0d6d1d9e061e92043d6d4cc81e3e82540c1cef

Based on the evidence presented in this paper, we demonstrate that blockchain implementations following Bitcoin's 2009 release are fundamentally flawed & therefore inferior & cannot deliver promised benefits. This calls for immediate regulatory intervention from global governments to protect consumers from potential financial harm. **Q.E.D**

10 Appendix

10.1 BoltNFT.sx

```
// BoltNFT.sx
import 'stdlib'
// Quick macro for reuse
#copyTopAlt fromAltStack dup toAltStack end
// Unlocking args
.ancestorPc1 .ancestorPc2 .ancestorPc3 .ancestorPc4 .ancestorPc5 .ancestorPc6
.fundOutpoint .changeOutput .beneficiaryPubKeyHash .sig .pubKey .ctx
| // Locking args & script
.pubKeyHash .pubKeyHashCommitment .txoType .parentOutpoint
.grandparentOutpoint .issuerPubKey .genesisOutpoint
not @label:isGenesis dup toAltStack
if 7n pick
    equalVerify // @Genesis pubKey MUST be issuer's to enforce sig
else drop // Use issuerPubKey from scriptCode
endIf
0notEqual @1:hasGrandparent swap drop // Use parentOutpoint from scriptCode
```

```
swap toAltStack // hasGrandparent
3n roll ifDup
if // If not melting
    dup 41 @1:sighashType verifyCtx // OP_PUSHTX
    // 2drop // Save compilation time
    splitCtx 5n roll autoSlice // Split ctx and slice scriptCode
    // Build the next token output(s)
    copyTopAlt notIf  // If prepareTx
        7n pick 14 24n roll cat tuck cat 0101 cat @1:tokenScript swap
        // build a bolt output
        0100000000000001976a9 swap cat 88ac cat @1:bolt swap
    else // if settleTx
        6n pick // must go beneficiary
        dup cat 0100 cat @1:tokenScript ff @1:offsetDummyValue swap // Hack ff byte
        // to keep stacks aligned
    endIf
    24 cat 15n pick cat // parentOutpoint
    6n pick cat // grandparentOutpoint
    4n roll @1:tokenScript // issuerPubKey
    // If mint/isGenesis
    fromAltStack fromAltStack dup rot swap toAltStack toAltStack
    if 24 16n pick cat // Build from ctx.outpoint
    else 4n pick // Copy genesisOutpoint direct from ctx
    endIf cat @1:tokenId // Unique blob
    3n roll // ctx.scriptCode
    cat @1:tokenIdScriptCode tuck // Duplicate
    cat @1:tokenScript
    size fd swap cat @1:scriptLength tuck // Duplicate
    0100000000000000 swap cat swap cat @1:token
    copyTopAlt notIf // If creating bolt
        24n roll // changeOutput
        4n roll swap cat cat // Concatenate bolt output
    else 25n roll cat // changeOutput
    endIf
    hash256 // Calculate hashOutputs
    copyTopAlt if 3n roll drop endIf // Drop ff offset byte
    11n roll equalVerify // ctx.hashOutputs check
    copyTopAlt 17n roll boolAnd @1:rebuildAncestor // txType && hasGrandparent
    copyTopAlt
    notIf 21n else 22n endIf
    roll swap tuck // Get fundOutpoint in place
    if // Rebuild ancestor
        // txVersion CAT txInsVi (02)
        28n roll 02 cat @1:grandparentTx // .ancestorPc1
        // vin outpoint + scriptVi + script to ctx up to & including outpoint CAT
        // scriptLength
```

```
        28n roll cat 3n pick cat @1:grandparentTx // .ancestorPc2
        // scriptCode data [pubKeyHash,pubKeyHashCommitment,txoType,parentOutpoint,
        // grandparentOutpoint,issuerPubKey,genesisOutpoint] CAT tokenIdScriptCode
        // @Genesis is 0000000000000000000
        27n roll cat @1:grandparentTx 4n pick 71n split nip cat @1:grandparentTx
        // .ancestorPc3 (Tx+1 ctx.genesisOutpoint is zeros)!
        // end of ctx (value to sighashType) + fund vin complete CAT txOutsVi (03) +
        // + 01000000 CAT scriptLength
        26n roll cat @1:grandparentTx 030100000000000000 cat 3n roll cat
        @1:grandparentTx // .ancestorPc4
        // scriptCode data [pubKeyHash,pubKeyHashCommitment,txoType,parentOutpoint,
        // grandparentOutpoint] CAT tokenIdScriptCode CAT bolt output
        24n roll cat 3n pick cat @1:grandparentTx 0100000000000001976a9 cat 9n roll 88ac
        cat cat @1:grandparentTx // .ancestorPc5
        // change output + nLockTime
        22n roll cat @1:grandparentTx
        hash256 @1:grandparentTxid 01000000 cat @1:grandparentOutpoint swap cat
        @1:boltAndFundOutpoints // .ancestorPc6
        12n // outpoint
    else 14n endIf // outpoint
    roll swap cat @1:prevOuts hash256 // Calculate hashPrevouts
    1n pick // rebuildAncestor
    if 13n else 15n endIf roll equalVerify // ctx.hashPrevouts check
endIf
if ff ff endif // Double stack alignment bytes :P
17n roll 17n roll dup hash160 18n roll equalVerify checkSig toAltStack
// Cleaning of stack to be removed when Teranode™ is released & consensus cares no more ;)
repeat 11n depth dup if 2n greaterThanOrEqual if 2drop else drop endif else drop endIf end
fromAltStack // CheckSig result
```

10.2 BoltNFT.sx.json

```
{
 "name": "boltNFT.sx",
 "compilerVersion": "0.1.0",
 "file": "contracts/boltNFT.sx",
 "unlockArgs": [
   "ancestorPc1",
   "ancestorPc2",
   "ancestorPc3",
   "ancestorPc4",
   "ancestorPc5",
   "ancestorPc6",
   "fundOutpoint",
   "changeOutput",
   "beneficiaryPubKeyHash",
   "sig",
   "pubKey",
   "ctx"
 ],
```

```
"lockArgs": [
  "pubKeyHash",
  "pubKeyHashCommitment",
  "txoType",
  "parentOutpoint",
  "grandparentOutpoint",
  "issuerPubKey",
  "genesisOutpoint"
],
"hex": "<pubKeyHash><pubKeyHashCommitment><txoType><parentOutpoint><grandparentOutpoint><issuerPub
↪ Key><genesisOutpoint>91766b63577988677568927c757c6b537a73637601416baa517f517f517f517f517f517f51
↪ 7f517f517f517f517f517f517f517f517f517f517f517f517f517f517f517f517f517f517f517f517f517f517f517f5
↪ 17f7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e
↪ 7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e01007e8100011f80517e9321414136d08c5ed2bf3ba048afe6dcaebafefffff
↪ ffffffffffffffffffffffff007d5296789f637897785296789f639467776867776876927f76927f76927f76927f76
↪ 927f76927f76927f76927f76927f76927f76927f76927f76927f76927f76927f76927f76927f76927f76927f76927f7
↪ 6927f76927f76927f76927f76927f76927f76927f76927f76927f76927f7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c
↪ 7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e7c7e8
↪ 27c7e23022079be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798027c7e827c7e01307c7e
↪ 6c7e2102b405d7f0322a89d0f9f3a98e6f938fdc1c969a8d1382a2bf66a71ae74a1e83b0ac69547f01207f01207f012
↪ 47f517f7c7601fd876375527f7c6701007e68817f587f547f01207f547f557a01157f01157f527f01257f01257f0122
↪ 7f01257f6c766b645779011401187a7e7d7e0201017e7c0b0100000000000000001976a97c7e0288ac7e7c675679767e0
↪ 201007e01ff7c6801247e5f797e56797e547a6c6c767b7c6b6b63012460797e675479687e537a7e7d7e8201fd7c7e7d
↪ 0801000000000000007c7e7c7e6c766b6401187a547a7c7e7e6701197a7e68aa6c766b63537a75685b7a886c766b011
↪ 17a9a6c766b640115670116687a7c7d63011c7a01027e011c7a7e53797e011b7a7e547901477f777e011a7a7e090301
↪ 000000000000007e537a7e01187a7e53797e0b0100000000000000001976a97e597a0288ac7e7e01167a7eaa040100000
↪ 07e7c7e5c675e687a7c7eaa5179635d675f687a88686301ff01ff6801117a01117a76a901127a88ac6b74766352a263
↪ 6d67756867756874766352a2636d67756867756874766352a2636d67756867756874766352a2636d677568677568747
↪ 66352a2636d67756867756874766352a2636d67756867756874766352a2636d67756867756874766352a2636d677568
↪ 67756874766352a2636d67756867756874766352a2636d67756867756874766352a2636d6775686775686c",
"asm": "<pubKeyHash> <pubKeyHashCommitment> <txoType> <parentOutpoint>
<grandparentOutpoint> <issuerPubKey> <genesisOutpoint> OP_NOT OP_DUP
OP_TOALTSTACK OP_IF OP_7 OP_PICK OP_EQUALVERIFY OP_ELSE OP_DROP OP_ENDIF
OP_0NOTEQUAL OP_SWAP OP_DROP OP_SWAP OP_TOALTSTACK OP_3 OP_ROLL OP_IFDUP
OP_IF OP_DUP 41 OP_TOALTSTACK OP_HASH256 OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT
OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE
OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT
OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE
OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT
OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE
OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT
OP_TRUE OP_SPLIT OP_TRUE OP_SPLIT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT
OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT
OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT
OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT
OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT
OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT
OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT 00 OP_CAT OP_BIN2NUM 0 1f
OP_NUM2BIN OP_1 OP_CAT OP_ADD
414136d08c5ed2bf3ba048afe6dcaebafeffffffffffffffffffffffffffffff00 OP_TUCK
OP_2 OP_DIV OP_OVER OP_LESSTHAN OP_IF OP_OVER OP_MOD OP_OVER OP_2 OP_DIV
OP_OVER OP_LESSTHAN OP_IF OP_SUB OP_ELSE OP_NIP OP_ENDIF OP_ELSE OP_NIP
OP_ENDIF OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP
OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL
OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP
OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL
OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP
OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL
```

```
OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP
OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL
OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP
OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL
OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP
OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_DUP OP_0NOTEQUAL
OP_SPLIT OP_DUP OP_0NOTEQUAL OP_SPLIT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP
OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP
OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP
OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP
OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP
OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP
OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SWAP OP_CAT OP_SIZE OP_SWAP OP_CAT
022079be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f8179802 OP_SWAP
OP_CAT OP_SIZE OP_SWAP OP_CAT 30 OP_SWAP OP_CAT OP_FROMALTSTACK OP_CAT
02b405d7f0322a89d0f9f3a98e6f938fdc1c969a8d1382a2bf66a71ae74a1e83b0 OP_CHECKSIG
OP_VERIFY OP_4 OP_SPLIT 20 OP_SPLIT 20 OP_SPLIT 24 OP_SPLIT OP_1 OP_SPLIT
OP_SWAP OP_DUP fd OP_EQUAL OP_IF OP_DROP OP_2 OP_SPLIT OP_SWAP OP_ELSE 00
OP_CAT OP_ENDIF OP_BIN2NUM OP_SPLIT OP_8 OP_SPLIT OP_4 OP_SPLIT 20 OP_SPLIT
OP_4 OP_SPLIT OP_5 OP_ROLL 15 OP_SPLIT 15 OP_SPLIT OP_2 OP_SPLIT 25 OP_SPLIT
25 OP_SPLIT 22 OP_SPLIT 25 OP_SPLIT OP_FROMALTSTACK OP_DUP OP_TOALTSTACK
OP_NOTIF OP_7 OP_PICK 14 18 OP_ROLL OP_CAT OP_TUCK OP_CAT 0101 OP_CAT OP_SWAP
01000000000000001976a9 OP_SWAP OP_CAT 88ac OP_CAT OP_SWAP OP_ELSE OP_6 OP_PICK
OP_DUP OP_CAT 0100 OP_CAT ff OP_SWAP OP_ENDIF 24 OP_CAT OP_15 OP_PICK OP_CAT
OP_6 OP_PICK OP_CAT OP_4 OP_ROLL OP_FROMALTSTACK OP_FROMALTSTACK OP_DUP OP_ROT
OP_SWAP OP_TOALTSTACK OP_TOALTSTACK OP_IF 24 OP_16 OP_PICK OP_CAT OP_ELSE OP_4
OP_PICK OP_ENDIF OP_CAT OP_3 OP_ROLL OP_CAT OP_TUCK OP_CAT OP_SIZE fd OP_SWAP
OP_CAT OP_TUCK 0100000000000000 OP_SWAP OP_CAT OP_SWAP OP_CAT OP_FROMALTSTACK
OP_DUP OP_TOALTSTACK OP_NOTIF 18 OP_ROLL OP_4 OP_ROLL OP_SWAP OP_CAT OP_CAT
OP_ELSE 19 OP_ROLL OP_CAT OP_ENDIF OP_HASH256 OP_FROMALTSTACK OP_DUP
OP_TOALTSTACK OP_IF OP_3 OP_ROLL OP_DROP OP_ENDIF OP_11 OP_ROLL OP_EQUALVERIFY
OP_FROMALTSTACK OP_DUP OP_TOALTSTACK 11 OP_ROLL OP_BOOLAND OP_FROMALTSTACK
OP_DUP OP_TOALTSTACK OP_NOTIF 15 OP_ELSE 16 OP_ENDIF OP_ROLL OP_SWAP OP_TUCK
OP_IF 1c OP_ROLL 02 OP_CAT 1c OP_ROLL OP_CAT OP_3 OP_PICK OP_CAT 1b OP_ROLL
OP_CAT OP_4 OP_PICK 47 OP_SPLIT OP_NIP OP_CAT 1a OP_ROLL OP_CAT
030100000000000000 OP_CAT OP_3 OP_ROLL OP_CAT 18 OP_ROLL OP_CAT OP_3 OP_PICK
OP_CAT 01000000000000001976a9 OP_CAT OP_9 OP_ROLL 88ac OP_CAT OP_CAT 16
OP_ROLL OP_CAT OP_HASH256 01000000 OP_CAT OP_SWAP OP_CAT OP_12 OP_ELSE OP_14
OP_ENDIF OP_ROLL OP_SWAP OP_CAT OP_HASH256 OP_1 OP_PICK OP_IF OP_13 OP_ELSE
OP_15 OP_ENDIF OP_ROLL OP_EQUALVERIFY OP_ENDIF OP_IF ff ff OP_ENDIF 11 OP_ROLL
11 OP_ROLL OP_DUP OP_HASH160 12 OP_ROLL OP_EQUALVERIFY OP_CHECKSIG
OP_TOALTSTACK OP_DEPTH OP_DUP OP_IF OP_2 OP_GREATERTHANOREQUAL OP_IF OP_2DROP
OP_ELSE OP_DROP OP_ENDIF OP_ELSE OP_DROP OP_ENDIF OP_DEPTH OP_DUP OP_IF OP_2
OP_GREATERTHANOREQUAL OP_IF OP_2DROP OP_ELSE OP_DROP OP_ENDIF OP_ELSE OP_DROP
OP_ENDIF OP_DEPTH OP_DUP OP_IF OP_2 OP_GREATERTHANOREQUAL OP_IF OP_2DROP
OP_ELSE OP_DROP OP_ENDIF OP_ELSE OP_DROP OP_ENDIF OP_DEPTH OP_DUP OP_IF OP_2
OP_GREATERTHANOREQUAL OP_IF OP_2DROP OP_ELSE OP_DROP OP_ENDIF OP_ELSE OP_DROP
OP_ENDIF OP_DEPTH OP_DUP OP_IF OP_2 OP_GREATERTHANOREQUAL OP_IF OP_2DROP
OP_ELSE OP_DROP OP_ENDIF OP_ELSE OP_DROP OP_ENDIF OP_DEPTH OP_DUP OP_IF OP_2
OP_GREATERTHANOREQUAL OP_IF OP_2DROP OP_ELSE OP_DROP OP_ENDIF OP_ELSE OP_DROP
OP_ENDIF OP_DEPTH OP_DUP OP_IF OP_2 OP_GREATERTHANOREQUAL OP_IF OP_2DROP
OP_ELSE OP_DROP OP_ENDIF OP_ELSE OP_DROP OP_ENDIF OP_DEPTH OP_DUP OP_IF OP_2
OP_GREATERTHANOREQUAL OP_IF OP_2DROP OP_ELSE OP_DROP OP_ENDIF OP_ELSE OP_DROP
OP_ENDIF OP_DEPTH OP_DUP OP_IF OP_2 OP_GREATERTHANOREQUAL OP_IF OP_2DROP
OP_ELSE OP_DROP OP_ENDIF OP_ELSE OP_DROP OP_ENDIF OP_FROMALTSTACK"
```

```
}
```

10.3 BoltFungible.sx

```
// BoltFungible.sx - BOLT Fungible
import 'stdlib'
// Unlocking args
.miscData // Miscellaneous data
.ancestorTxAVersion
.ancestorTxAVin1Header // Outpoint + ScriptVi
.ancestorTxAVin1MiscData
.ancestorTxAVin1DataToCTX // Pop ancestor data off
.ancestorTxAVin1CTXToScriptCode
.ancestorTxAVin1CTXScriptCodeData // Pop issuerPubKey off and check
.ancestorTxAVin1CTXEndNSequence // Pop nSequence off
.ancestorTxAVin2Header // Outpoint + ScriptVi
.ancestorTxAVin2MiscData
.ancestorTxAVin2DataToCTX // Pop ancestor data off
.ancestorTxAVin2CTXToScriptCode
.ancestorTxAVin2CTXScriptCodeData // Pop issuerPubKey off and check
.ancestorTxAVin2CTXEndNSequence // Pop nSequence off
.ancestorTxAVinFundVin
.ancestorTxAVout1Data // Check issuerPubKey
.ancestorTxAVout2Data // Check issuerPubKey
.ancestorTxABoltPubKeyHash1
.ancestorTxABoltPubKeyHash2
.ancestorTxAChangeNLockTime
// 2nd Ancestor for Merge & Swap
.ancestorTxBVersion
.ancestorTxBVin1Header // Outpoint + ScriptVi
.ancestorTxBVin1MiscData
.ancestorTxBVin1DataToCTX // Pop ancestor data off
.ancestorTxBVin1CTXToScriptCode
.ancestorTxBVin1CTXScriptCodeData // Pop issuerPubKey off and check
.ancestorTxBVin1CTXEndNSequence // Pop nSequence off
.ancestorTxBVin2Header // Outpoint + ScriptVi
.ancestorTxBVin2MiscData
.ancestorTxBVin2DataToCTX // Pop ancestor data off
.ancestorTxBVin2CTXToScriptCode
.ancestorTxBVin2CTXScriptCodeData // Pop issuerPubKey off and check
.ancestorTxBVin2CTXEndNSequence // Pop nSequence off
.ancestorTxBVinFundVin
.ancestorTxBVout1Data // Check issuerPubKey
.ancestorTxBVout2Data // Check issuerPubKey
.ancestorTxBBoltPubKeyHash1
.ancestorTxBBoltPubKeyHash2
```

```
.ancestorTxBChangeNLockTime
// More
.grandparentBoltVoutIdx // Validate TXID against parents (need to know what bolt) 4 bytes
.interopBoltVoutIdx // Validate TXID against parents (need to know what bolt) 4 bytes
.interopPubKeyHash // Needed for second token
.interopOutpoint // Immediate paired outpoint for hashPrevout calc
.interopParentOutpoint // Ditto
.interopGenesisOutpoint // Bottom token needs new
.interopIssuerPubKey // Bottom token needs new
.fundOutpoint // The input funding the tx's outpoint
.changeOutput // The output of leftover sats from above input
.pubKeyHash1 // Beneficiary OR retrospective dual bolt tx rebuild
.pubKeyHash2 // Beneficiary OR retrospective dual bolt tx rebuild
.nextBalanceCommit // The amount of balance to be split or merged with
.nextTxoType // Action byte [20-27]:transferSettle,transferCommit,splitSettle,splitCommit,mergeSett
.inputIndexN // 1 byte (needed for swap)
.sig // Standard P2PKH
.pubKey // Standard P2PKH
.ctx // ScriptContext (SigHashPreimage)
| // Locking args
.mintData
.balance // Current balance
.balanceCommit // Next differential commit
.pubKeyHash
.pubKeyHashCommit
.pubKeyHashCommit2 // other commitment for split/swap
.otherParentOutpoint
.otherGrandparentOutpoint
.otherGenesisOutpoint
.otherIssuerPubKey
.txoType // Current token mode/status
.outputIndexN // All 00 at mint regardless
.parentOutpoint
.grandparentOutpoint
.genesisOutpoint
.issuerPubKey

// Logic Script
swap bin2num not @label:isGenesis tuck
if 17n pick /*pubKey*/ equalVerify // @Genesis pubKey MUST be issuer's
    19n pick /*nextTxoType*/ dup 25 equal not verify 27 equal not verify // You cannot merge or sw
else drop // Keep stacks aligned (use scriptCode.issuerPubKey)
endIf
swap dup bin2num 0notEqual @1:hasGrandparent 16n pick /*ctx*/ ifDup
if // If not melting
    dup 41 @1:sighashType // verifyCtx
```

```
    2drop // Save compilation time
    splitCtx // Split ctx and slice scriptCode
    5n roll /*ctx.scriptCode*/
    // SKIPPING MINT DATA
    25n pick /*mintData*/ size nip dup
    4b lessThanOrEqual if 1n else dup
    4c lessThanOrEqual if 2n else dup // OP_PUSHDATA1
    4d lessThanOrEqual if 3n else dup // OP_PUSHDATA2
    4e equal if 5n endIf endIf endIf endIf add // OP_PUSHDATA4
    split @l:scriptCode.mintData

    // Split the rest of the script args with their known lengths
    9n split @l:scriptCode.balance
    9n split @l:scriptCode.balanceCommit
    21n split @l:scriptCode.pubKeyHash
    21n split @l:scriptCode.pubKeyHashCommit
    21n split @l:scriptCode.pubKeyHashCommit2
    37n split @l:scriptCode.otherParentOutpoint
    37n split @l:scriptCode.otherGrandparentOutpoint
    37n split @l:scriptCode.otherGenesisOutpoint
    34n split @l:scriptCode.otherIssuerPubKey
    2n split @l:scriptCode.txoType
    1n split @l:scriptCode.outputIndexN
    37n split @l:scriptCode.parentOutpoint
    37n split @l:scriptCode.grandparentOutpoint
    37n split @l:scriptCode.genesisOutpoint
    34n split @l:scriptCode.issuerPubKey
    /*** Ensure protocol settleTx follows commitTx or visa versa ***/
    31n pick /*txoType*/ 2n mod @l:hasBolt dup
    if // If we have a bolt the nextTxoType must be the equivalent settle txoType
        47n pick /*nextTxoType*/ 1add 33n pick /*txoType*/ equalVerify
    endIf

    // Check nextTxoType is valid (e.g. opposite of hasBolt)
    dup 48n pick /*nextTxoType*/ 2n mod swap if 0n else 1n endIf equalVerify

    // Define reusable values
    1d02b0178876a914 @l:boltHeader
    fffffffffff1f00 @l:tempMaxBalance // Javascript.MAX_SAFE_INTEGER
    08000000000000000 @l:nullBalance
    140000000000000000000000000000000000000000 @l:nullPubKeyHash
    2100000000000000000000000000000000000000000000000000000000000000000000000000 @l:nullPubKey
    240000000000000000000000000000000000000000000000000000000000000000000000000000000000 @l:nullOutpoint

    // Enforce balance rules
    47n pick /*balance*/ dup 0n greaterThanOrEqual verify 5n pick /*tempMaxBalance*/ lessThanOrEqu
```

```
135     54n pick /*nextBalanceCommit*/ dup 0n greaterThanOrEqual verify 5n pick /*tempMaxBalance*/ les
136
137     // Handy booleans
138     53n pick /*nextTxoType*/
139     dup 20 equal @1:isTransferSettle swap
140     dup 21 equal @1:isTransferCommit swap
141     dup 22 equal @1:isSplitSettle swap
142     dup 23 equal @1:isSplitCommit swap
143     dup 24 equal @1:isMergeSettle swap
144     dup 25 equal @1:isMergeCommit swap
145     dup 26 equal @1:isSwapSettle swap
146     27 equal @1:isSwapCommit
147
148     // Ensure single satoshi
149     36n pick /*ctx.value*/ bin2num 1n equalVerify
150
151     // Further balance checks
152     // If merging ensure balance added is less than MAX & addition is positive
153     2n pick /*isMergeCommit*/
154     if 55n pick /*balance*/ 63n pick /*nextBalanceCommit*/ add 13n pick /*tempMaxBalance*/ lessTh
155     // If splitting ensure balance deducted is greater than or equal to ZERO & balance
156     4n pick /*isSplitCommit*/
157     if 55n pick /*balance*/ 63n pick /*nextBalanceCommit*/
158         2dup sub 0n greaterThanOrEqual verify greaterThanOrEqual verify
159     endIf
160
161     // Build bolt(s)
162     14n pick /*hasBolt*/
163     notIf // If spending from a settle we must commit a bolt
164         36n pick /*ctx.value*/ 14n pick /*boltHeader*/ cat dup
165         66n pick /*pubKeyHash1*/ cat 88ac cat @1:bolt1
166         6n pick /*isSplitCommit*/ 3n pick /*isSwapCommit*/ boolOr if // Build a 2nd bolt
167             swap 65n pick /*pubKeyHash2*/ cat 88ac cat @1:bolt2
168             62n pick /*inputIndexN*/ if swap endIf
169             cat @1:bolts
170         else nip endIf
171     else ff @1:alignByte // A byte to keep commit/settle tx stack indices in sync
172     endIf
173     // So we have our bolt(s)
174     // Now we build the token output(s)
175     02b017 /*mintData*/
176     // First we start with the balances
177     8n pick /*isTransferCommit*/ 10n pick /*isTransferSettle*/ boolOr 4n pick /*isSwapSettle*/ bo
178         32n pick /*scriptCode.balance*/ cat 13n pick /*nullBalance*/
179     else
180         2n pick /*isSwapCommit*/ 5n pick /*isMergeCommit*/ boolOr 7n pick /*isSplitCommit*/ boolO
```

```
            32n pick /*scriptCode.balance*/ 08 66n pick /*nextBalanceCommit*/ cat
            6n pick /*isMergeCommit*/ 65n pick /*inputIndexN*/ boolAnd if swap endIf
            cat
        else
            5n pick /*isMergeSettle*/ if
                08 58n pick /*balance*/ 58n pick /*balanceCommit*/ add
                repeat 8n size 8n lessThan if 00 cat endIf end // pad the balance
                cat 14n pick /*nullBalance*/ cat
            else
                7n pick /*isSplitSettle*/ if
                    08 58n pick /*balance*/ 58n pick /*balanceCommit*/ sub
                    repeat 8n size 8n lessThan if 00 cat endIf end // pad the balance
                    cat 14n pick /*nullBalance*/ cat
                endIf
            endIf
        endIf
    endIf cat

    // Now pubKeyHashes
    8n pick /*isTransferCommit*/ if
        30n pick /*scriptCode.pubKeyHash*/ cat 14 67n pick /*pubKeyHash1*/ cat cat 12n pick /*nul
    else 9n pick /*isTransferSettle*/ 6n pick /*isMergeSettle*/ boolOr 8n pick /*isSplitSettle*/
        29n pick /*scriptCode.pubKeyHashCommit*/ cat 12n pick dup /*nullPubKeyHash*/ cat
        else
            2n pick /*isSwapCommit*/ 7n pick /*isSplitCommit*/ boolOr if
                30n pick /*scriptCode.pubKeyHash*/ cat 14 67n pick /*pubKeyHash1*/ cat 14 67n pick
            else
                3n pick /*isSwapSettle*/ if
                    28n pick /*scriptCode.pubKeyHashCommit2*/ cat 12n pick dup /*nullPubKeyHash*/
                else
                    4n pick /*isMergeCommit*/ if
                        62n pick /*inputIndexN*/ notIf
                            30n pick /*scriptCode.pubKeyHash*/
                            else
                                14 74n pick /*interopPubKeyHash*/ cat
                            endIf
                        cat 14 67n pick /*pubKeyHash1*/ cat cat 12n pick /*nullPubKeyHash*/
                    endIf
                endIf
            endIf
        endIf
    endIf cat

    // Now future merge/swap commit args
    8n pick /*isTransferCommit*/ 10n pick /*isTransferSettle*/ boolOr 4n pick /*isSwapSettle*/ bo
        10n pick /*nullOutpoint*/ cat
```

```
        10n pick /*nullOutpoint*/ cat
        10n pick /*nullOutpoint*/ cat
        11n pick /*nullPubKey*/ cat
    else
        2n pick /*isSwapCommit*/ if
            24 73n pick /*interopOutpoint*/ cat cat
            24 72n pick /*interopParentOutpoint*/ cat cat
            24 71n pick /*interopGenesisOutpoint*/ cat cat
            21 70n pick /*interopIssuerPubKey*/ cat cat
        else
            4n pick /*isMergeCommit*/ if
                62n pick /*inputIndexN*/ notIf
                    24 73n pick /*interopOutpoint*/ cat cat
                    24 72n pick /*interopParentOutpoint*/ cat cat
                else
                    24 40n pick /*ctx.outpoint*/ cat cat
                    21n pick /*scriptCode.parentOutpoint*/ cat
                endIf
                10n pick /*nullOutpoint*/ cat
                11n pick /*nullPubKey*/ cat
            endIf
        endIf
    endIf
    // Common
    01 64n pick /*nextTxoType*/ cat cat
    4n pick /*isMergeCommit*/ 63n pick /*inputIndexN*/ boolAnd if
        00 cat // 2nd token > 1st /*outputIndexN*/
        24 73n pick /*interopOutpoint*/ cat cat
        24 72n pick /*interopParentOutpoint*/ cat cat
        24 71n pick /*interopGenesisOutpoint*/ cat cat
    else
        3n pick /*isSwapSettle*/ if 47n /*outputIndexN*/ else 62n endIf pick /*inputIndexN*/
        notIf 00 else 51 endIf cat /*outputIndexN*/
        24 40n pick /*ctx.outpoint*/ cat cat
        21n pick /*scriptCode.parentOutpoint*/ cat
        45n pick /*isGenesis*/ if
            24 40n pick /*ctx.outpoint*/ cat else 19n pick /*scriptCode.genesisOutpoint*/
        endIf cat
    endIf
    18n pick /*scriptCode.issuerPubKey*/ cat
    17n pick /*ctx.scriptCode*/ cat
    // Append serialisation length
    size fd swap cat @1:scriptLength tuck
    40n pick /*ctx.value*/
    swap cat swap cat @1:token

```

```
273     // Now we have to build the 2nd token output if splitSettle or swapping
274     3n pick /*isSwapCommit*/ 5n pick /*isSwapSettle*/ boolOr 9n pick /*isSplitSettle*/ boolOr if
275         02b017 /*mintData*/
276         // First we start with the balances
277         4n pick /*isSwapCommit*/ if
278             08 67n pick /*nextBalanceCommit*/ cat 35n pick /*scriptCode.balance*/ cat cat
279             // Now pubKeyHashes
280             14 76n pick /*interopPubKeyHash*/ cat cat 14 68n pick /*pubKeyHash2*/ cat 14 70n pick
281             // Now future merge/swap commit args
282             24 42n pick /*ctx.outpoint*/ cat cat
283             23n pick /*scriptCode.parentOutpoint*/ cat
284             21n pick /*scriptCode.genesisOutpoint*/ cat
285             20n pick /*scriptCode.issuerPubKey*/ cat
286         else
287             33n pick /*scriptCode.balanceCommit*/ cat 15n pick /*nullBalance*/ cat
288             9n pick /*isSplitSettle*/ notIf
289             31n else 30n endIf pick /*scriptCode.pubKeyHashCommit/pubKeyHashCommit2*/ cat 14n pick
290             // other vars are null
291             12n pick /*nullOutpoint*/ cat
292             12n pick /*nullOutpoint*/ cat
293             12n pick /*nullOutpoint*/ cat
294             13n pick /*nullPubKey*/ cat
295         endIf
296         // Common
297         01 66n pick /*nextTxoType*/ cat cat 4n pick /*isSwapCommit*/ if 64n /*inputIndexN*/ else
298
299         5n pick /*isSwapSettle*/ if
300             // Build outpoint from our own
301             24 42n pick /*ctx.outpoint*/ 32n split not // invert
302             if 01000000 else 00000000 endIf cat @l:otherInOutpoint dup toAltStack cat cat // padding
303             29n pick /*scriptCode.otherParentOutpoint*/ cat
304             27n pick /*scriptCode.otherGenesisOutpoint*/ cat
305             26n pick /*scriptCode.otherIssuerPubKey*/ cat
306         else
307             9n pick /*isSplitSettle*/ if
308                 24 42n pick /*ctx.outpoint*/ cat cat // parentOutpoint
309                 23n pick /*scriptCode.parentOutpoint*/ cat
310                 21n pick /*scriptCode.genesisOutpoint*/ cat
311                 20n pick /*scriptCode.issuerPubKey*/ cat
312             else
313                 24 75n pick /*interopOutpoint*/ cat cat
314                 24 74n pick /*interopParentOutpoint*/ cat cat
315                 24 73n pick /*interopGenesisOutpoint*/ cat cat
316                 21 72n pick /*interopIssuerPubKey*/ cat cat
317             endIf
318         endIf
```

```
319         19n pick /*ctx.scriptCode*/ cat
320         // Append serialisation length
321         // size fd swap cat @l:scriptLength tuck
322         2n pick /*scriptLength*/
323         41n pick /*ctx.value*/
324         swap cat swap cat @l:token
325         4n pick /*isSwapCommit*/ if
326         64n /*inputIndexN*/ else 49n /*outputIndexN*/ endIf pick if swap endIf
327         cat @l:tokens
328     endIf
329
330     2n roll /*bolt1/bolts/alignByte*/
331     // Check if we are creating a bolt
332     17n pick /*hasBolt*/ notIf // If commitTx
333         cat // Append serialised bolt
334         else drop
335     endIf
336     // Add the change output
337     67n pick /*changeOutput*/ cat @l:serialisedOutputs
338     // Check hashOutputs
339     hash256 36n pick /*ctx.hashOutputs*/ equalVerify
340     // Check input outpoints
341     38n pick /*ctx.outpoint*/
342     2n pick /*isSwapCommit*/ 5n pick /*isMergeCommit*/ boolOr if
343         72n pick /*interopOutpoint*/ 63n pick /*inputIndexN*/ if swap endIf cat @l:tokenPrevOuts
344     else 3n pick /*isSwapSettle*/ if
345         fromAltStack 48n pick /*outputIndexN*/ if swap endIf cat @l:tokenPrevOuts
346         endIf
347     endIf
348     16n pick /*hasBolt*/ 44n pick /*hasGrandparent*/ boolAnd @l:rebuildAncestor
349     /*
350     txVersion
351     txInsVi (KNOWN)
352         txHashBuf & txOutNum
353         scriptVi & script
354         scriptData to ctx.outpoint
355             ctx.scriptCode.scriptVi (KNOWN)
356             ctx.scriptCode.data to genesisOutpoint
357             ctx.scriptCode
358         ctx.end & nSequence
359     txOutsVi
360         valueBn
361         scriptVi & script
362         scriptData to genesisOutpoint
363         scriptCode
364     nLocktime
```

```
365     */
366     if
367         dup 01 equal @l:ancestor1IsTransferCommit swap // Vins:T,F Vouts:T,B,C
368         dup 03 equal @l:ancestor1IsSplitCommit swap // Vins:T,F Vouts:T,B,B,C
369         dup 05 equal @l:ancestor1IsMergeCommit swap // Vins:T,T,F Vouts:T,B,C
370         07 equal @l:ancestor1IsSwapCommit // T,T,F Vouts:T,T,B,B,C*/
371         113n pick /* ancestorTxAVersion */ 107n pick /* ancestorTxAVin2Header */ size
372         0n greaterThan @l:ancestorHas2TokenInputs nip dup if 3n else 2n endIf
373         2n roll swap cat
374         114n pick /* ancestorTxAVin1Header */ cat 113n pick /* ancestorTxAVin1MiscData */
375         size dup // WritePushData
376         // !!! This needs proper testing !!!
377         4b lessThanOrEqual if else dup // Do nothing
378         4c lessThanOrEqual if 4c swap 1n else dup // OP_PUSHDATA1
379         4d lessThanOrEqual if 4d swap 2n else dup // OP_PUSHDATA2
380         4e swap 4n endIf endIf num2bin cat endIf // OP_PUSHDATA4
381         // !!! This needs proper testing !!!
382         swap cat cat
383         112n pick /* ancestorTxAVin1DataToCTX */ cat
384         111n pick /* ancestorTxAVin1CTXToScriptCode */ cat
385         110n pick /* ancestorTxAVin1CTXScriptCodeData */
386         20n pick /* ctx.scriptCode */ cat
387         // Script is never less than fd
388         // TEST THIS
389         size dup ffff bin2num swap lessThanOrEqual if
390             fd swap else
391                 dup ffffffff bin2num swap lessThanOrEqual if
392                     fe swap size 3n equal if 00 swap cat endIf // 2 bytes covered above, if 3 pad to
393                 else
394                     ff swap repeat 3n size 8n lessThan if 00 swap cat endIf end // 4 bytes covered a
395                 endIf
396             endIf cat swap cat cat
397         109n pick /* ancestorTxAVin1CTXEndNSequence */ cat
398          // Build 2nd token input
399         swap if
400                 107n pick /* ancestorTxAVin2Header */ cat 106n pick /* ancestorTxAVin2MiscData */
401                 size dup // WritePushData
402                 // !!! This needs proper testing !!!
403                 4b lessThanOrEqual if else dup // Do nothing
404                 4c lessThanOrEqual if 4c swap 1n else dup // OP_PUSHDATA1
405                 4d lessThanOrEqual if 4d swap 2n else dup // OP_PUSHDATA2
406                 4e swap 4n endIf endIf num2bin cat endIf // OP_PUSHDATA4
407                 // !!! This needs proper testing !!!
408                 swap cat cat
409                 105n pick /* ancestorTxAVin2DataToCTX */ cat
410                 104n pick /* ancestorTxAVin2CTXToScriptCode */ cat
```

```
411                 103n pick /* ancestorTxAVin2CTXScriptCodeData */
412                 19n pick /* ctx.scriptCode */ cat
413                 // Script is never less than fd
414                 // TEST THIS
415                 size dup ffff bin2num swap lessThanOrEqual if
416                     fd swap else
417                         dup ffffffff bin2num swap lessThanOrEqual if
418                             fe swap size 3n equal if 00 swap cat endIf // 2 bytes covered above, if
419                         else
420                             ff swap repeat 3n size 8n lessThan if 00 swap cat endIf end // 4 bytes
421                         endIf
422                     endIf cat swap cat cat
423                     102n pick /* ancestorTxAVin2CTXEndNSequence */ cat
424             endIf
425             101n pick /* ancestorTxAVinFundVin */ cat
426             // Now outputs
427             99n pick /* ancestorTxAVout2Data */ size 0n greaterThan @1:ancestorHas2TokenOutputs nip
428             98n pick /* ancestorTxABoltPubKeyHash2 */ size 0n greaterThan @1:ancestorHas2Bolts nip
429             2dup boolAnd if 5n else 2dup boolOr if 4n else 3n endIf endIf 3n roll swap cat // txOuts
430             41n pick /* ctx.value */ cat 4n pick /* scriptLength */ cat 102n pick /* ancestorTxAVout1D
431             2n roll /* ancestorHas2TokenOutputs */ if
432                     40n pick /* ctx.value */ cat 3n pick /* scriptLength */ cat 100n pick /* ancestorT
433             endIf
434             // Now the bolt(s)
435             40n pick /* ctx.value */ cat 17n pick /* boltHeader */ cat 99n pick /* ancestorTxABoltPubK
436             swap if // Build 2nd bolt
437                 39n pick /* ctx.value */ cat 16n pick /* boltHeader */ cat 97n pick /* ancestorTxABolt
438             endIf
439             96n pick /* ancestorTxAChangeNLockTime */ cat @1:grandparentATx
440             // Verify grandparent
441             hash256 dup 46n pick /* grandparentOutpoint */ 32n split drop equalVerify
442             76n pick /* grandparentBoltVoutIdx */ cat @1:grandparentBoltOutpoint
443
444             // Now we check if we have to build 2nd ancestor
445             6n pick /* isMergeSettle */ 5n pick /* isSwapSettle */ boolOr if
446                 // Build 2nd ancestor
447                 95n pick /* ancestorTxBVersion */ 89n pick /* ancestorTxBVin2Header */ size
448                 0n greaterThan @1:ancestorHas2TokenInputs nip dup if 3n else 2n endIf
449                 2n roll swap cat
450                 96n pick /* ancestorTxBVin1Header */
451                 cat 95n pick /* ancestorTxBVin1MiscData */
452                 size dup // WritePushData
453                 // !!! This needs proper testing !!!
454                 4b lessThanOrEqual if else dup // Do nothing
455                 4c lessThanOrEqual if 4c swap 1n else dup // OP_PUSHDATA1
456                 4d lessThanOrEqual if 4d swap 2n else dup // OP_PUSHDATA2
```

```
457                 4e swap 4n endIf endIf num2bin cat endIf // OP_PUSHDATA4
458                 // !!! This needs proper testing !!!
459                 swap cat cat
460                 94n pick /* ancestorTxBVin1DataToCTX */ cat
461                 93n pick /* ancestorTxBVin1CTXToScriptCode */ cat
462                 92n pick /* ancestorTxBVin1CTXScriptCodeData */
463                 21n pick /* ctx.scriptCode */ cat
464                 // false verify
465                 // Script is never less than fd
466                 // TEST THIS
467                 size dup ffff bin2num swap lessThanOrEqual if
468                     fd swap else
469                     dup ffffffff bin2num swap lessThanOrEqual if
470                         fe swap size 3n equal if 00 swap cat endIf // 2 bytes covered above, if 3 p
471                     else
472                         ff swap repeat 3n size 8n lessThan if 00 swap cat endIf end // 4 bytes cove
473                     endIf
474                 endIf cat swap cat cat
475                 91n pick /* ancestorTxBVin1CTXEndNSequence */ cat
476                 // Build 2nd token input
477                 swap if
478                         89n pick /* ancestorTxBVin2Header */ cat 88n pick /* ancestorTxBVin2MiscData *
479                         size dup // WritePushData
480                         // !!! This needs proper testing !!!
481                         4b lessThanOrEqual if else dup // Do nothing
482                         4c lessThanOrEqual if 4c swap 1n else dup // OP_PUSHDATA1
483                         4d lessThanOrEqual if 4d swap 2n else dup // OP_PUSHDATA2
484                         4e swap 4n endIf endIf num2bin cat endIf // OP_PUSHDATA4
485                         // !!! This needs proper testing !!!
486                         swap cat cat
487                         87n pick /* ancestorTxBVin2DataToCTX */ cat
488                         86n pick /* ancestorTxBVin2CTXToScriptCode */ cat
489                         85n pick /* ancestorTxBVin2CTXScriptCodeData */
490                         19n pick /* ctx.scriptCode */ cat
491                         // Script is never less than fd
492                         // TEST THIS
493                         size dup ffff bin2num swap lessThanOrEqual if
494                             fd swap else
495                                 dup ffffffff bin2num swap lessThanOrEqual if
496                                     fe swap size 3n equal if 00 swap cat endIf // 2 bytes covered a
497                                 else
498                                     ff swap repeat 3n size 8n lessThan if 00 swap cat endIf end //
499                                 endIf
500                             endIf cat swap cat cat
501                         84n pick /* ancestorTxBVin2CTXEndNSequence */ cat
502                 endIf
```

```
                83n pick /*ancestorTxBVinFundVin*/ cat
                // Now outputs
                81n pick /*ancestorTxBVout2Data*/ size 0n greaterThan @1:ancestorHas2TokenOutputs n
                80n pick /*ancestorTxBBoltPubKeyHash2*/ size 0n greaterThan @1:ancestorHas2Bolts ni
                2dup boolAnd if 5n else 2dup boolOr if 4n else 3n endIf endIf 3n roll swap cat // ta
                42n pick /*ctx.value*/ cat 5n pick /*scriptLength*/ cat 84n pick /*ancestorTxBVou
                2n roll /*ancestorHas2TokenOutputs*/ if
                    41n pick /*ctx.value*/ cat 4n pick /*scriptLength*/ cat 82n pick /*ancestorTx
                endIf
                // false verify
                // Now the bolt(s)
                41n pick /*ctx.value*/ cat 18n pick /*boltHeader*/ cat 81n pick /*ancestorTxBBolt
                swap if // Build 2nd bolt
                    40n pick /*ctx.value*/ cat 17n pick /*boltHeader*/ cat 79n pick /*ancestorTxB
                endIf
                78n pick /*ancestorTxBChangeNLockTime*/ cat @1:grandparentBTx
                // Verify grandparent
                hash256 dup 54n pick /*otherGrandparentOutpoint*/ 32n split drop equalVerify
                76n pick /*interopBoltVoutIdx*/ cat @1:otherBoltOutpoint
                49n pick /*outputIndexN*/ if swap endIf cat
        endIf
        cat
    else
        // Ensure all ancestor variables are null length
        repeat 38n 113n pick end repeat 37n cat end size nip 0n equalVerify
    endIf
    // Concatenate the funding input outpoint
    68n pick /*fundOutpoint*/ cat @1:prevOuts
    // Check hashPrevouts
    hash256 41n pick /*ctx.hashPrevouts*/ equalVerify
endif
```

10.4 BoltFungible.sx.json

```
{
  {
    "name": "BoltFungible.sx",
    "compilerVersion": "0.1.0",
    "file": "./sx/contracts/bolt/boltFungible.sx",
    "lockHex": "<mintData><balance><balanceCommit><pubKeyHash><pubKeyHashCommit><pubKeyHashCommit2>
↪  <otherParentOutpoint>
    <otherGrandparentOutpoint><otherGenesisOutpoint><otherIssuerPubKey><txoType><outputIndexN><pare
↪  ntOutpoint>
    <grandparentOutpoint><genesisOutpoint><issuerPubKey>7c81917d6301117988011379760125879169012787 9
↪  1696775687c7681926079
    73637601416d547f01207f01207f01247f517f7c7601fd876375527f7c6701007e68817f587f547f01207f547f557a0
↪  11979827776014ba16351
    6776014ca163526776014da163536776014e876355686868937f597f597f01157f01157f01157f01257f01257f012
↪  57f01227f527f517f0125
```

```
7f01257f01257f01227f011f7952977663012f798b012179886876013079529 77c630067516888081d02b0178876a91
↪ 408fffffffffff1f0009
0800000000000000001514000000000000000000000000000000000000000000222100000000000000000000000000000000
↪ 00000000000000000000
0000000000000002524000000000000000000000000000000000000000000000000000000000000000000000000000000012
↪ f797600a2695579a16901
36797600a2695579a169013579760120877c760121877c760122877c760123877c760124877c760125877c760126877
↪ c01278701247981518852
7963013779013f79935d79a1696854796301377901 3f796e9400a269a269685e79640124795e797e760142797e0288a
↪ c7e567953799b637c0141
797e0288ac7e013e79637c687e6777686701ff680302b01758795a799b54799b630120797e5d7967527955799b57799
↪ b6301207901080142797e
56790141799a637c687e675579630108013a79013a799382589f6301007e6882589f6301007e6882589f6301007e688
↪ 2589f6301007e6882589f
6301007e6882589f6301007e6882589f6301007e6882589f6301007e687e5e797e675779630108013a79013a7994825
↪ 89f6301007e6882589f63
01007e6882589f6301007e6882589f6301007e6882589f6301007e6882589f6301007e6882589f6301007e6882589f6
↪ 301007e687e5e797e6868
68687e587963011e797e01140143797e7e5c7967597956799b58799b63011d797e5c79767e67527957799b63011e797
↪ e01140143797e01140143
797e7e67537963011c797e5c79767e67547963013e7964011e79670114014a797e687e01140143797e7e5c796868686
↪ 8687e58795a799b54799b
56799b57799b58799b635a797e5a797e5a797e5b797e675279630124014979 7e7e01240148797e7e01240147797e7e0
↪ 1210146797e7e67547963
013e796401240149797e7e01240148797e7e6701240128797e7e0115797e685a797e5b797e68686801010140797e7e5
↪ 479013f799a6301007e01
240149797e7e01240148797e7e01240147797e7e67537963012f67013e6879640100670151687e01240128797e7e011
↪ 5797e012d796301240128
797e67011379687e680112797e0111797e8201fd7c7e7d0128797c7e7c7e537955799b59799b630302b017547963010
↪ 80143797e0123797e7e01
14014c797e7e01140144797e01140146797e7e7e0124012a797e7e0117797e0115797e0114797e670121797e5f797e5
↪ 97964011f67011e68797e
5e79767e7e5c797e5c797e5c797e5d797e6801010142797e7e5479630140670131687964015167010068 7e557963012
↪ 4012a7901207f91630401
```

```
    0000006704000000000687e766b7e7e011d797e011b797e011a797e675979630124012a797e7e0117797e0115797e011
↪   4797e670124014b797e7e0124014a797e7e01240149797e7e01210148797e7e68680113797e52790129797c7e7c7e54
↪   7963014067013168796376c687e68527a011179647e6775680143797eaa0124798801267952795579 9b63014879013f7
↪   9637c687e675379636c013079637c687e68686079012c799a63760101877c760103877c760105877c01078701717901
↪   6b798200a077766353675268527a7c7e0172797e0171798276014ba1636776014ca163014c7c516776014da163014d7
↪   c526776014e7c546868807e687c7e7e0170797e016f797e016e790114797e827602ffff817ca16301fd7c677604ffff
↪   ffff817ca16301fe7c8253876301007c7e686701ff7c82589f6301007c7e6882589f6301007c7e6882589f6301007c7
↪   e6868687e7c7e7e016d797e7c63016b797e016a798276014ba1636776014ca163014c7c516776014da163014d7c5267
↪   76014e7c546868807e687c7e7e0169797e0168797e0167790113797e827602ffff817ca16301fd7c677604ffffffff8
↪   17ca16301fe7c8253876301007c7e686701ff7c82589f6301007c7e6882589f6301007c7e6882589f6301007c7e6868
↪   687e7c7e7e0166797e680165797e0163798200a0770162798200a0776e9a6355676e9b635467536868537a7c7e01297
↪   97e54797e0166797e0114797e527a630128797e53797e0164797e0113797e680128797e0111797e0163797e0288ac7e
↪   7c630127797e60797e0161797e0288ac7e680160797eaa76012e7901207f7588014c797e567955799b63015f7901597
↪   98200a077766353675268527a7c7e0160797e015f798276014ba1636776014ca163014c7c516776014da163014d7c52
↪   6776014e7c546868807e687c7e7e015e797e015d797e015c790115797e827602ffff817ca16301fd7c677604ffffffff
↪   f817ca16301fe7c8253876301007c7e686701ff7c82589f6301007c7e6882589f6301007c7e6882589f6301007c7e68
↪   68687e7c7e7e015b797e7c630159797e0158798276014ba1636776014ca163014c7c516776014da163014d7c5267760
↪   14e7c546868807e687c7e7e0157797e0156797e0155790113797e827602ffff817ca16301fd7c677604ffffffff817c
↪   a16301fe7c8253876301007c7e686701ff7c82589f6301007c7e6882589f6301007c7e6882589f6301007c7e6868687
↪   e7c7e7e0154797e680153797e0151798200a0770150798200a0776e9a6355676e9b635467536868537a7c7e012a797e
↪   55797e0154797e0115797e527a630129797e54797e0152797e0114797e680129797e0112797e0151797e0288ac7e7c6
↪   30128797e0111797e014f797e0288ac7e68014e797eaa7601367901207f7588014c797e013179637c687e687e670171
↪   790171790171790171790171790171790171790171790171790171790171790171790171790171790171790171790171
↪   179017179017179017179017179017179017179017179017179017179017179017179017179017179017179017179017
↪   7179017179017179017179017179017171797e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7
↪   e7e7e7e7e7e7e827700886801447 97eaa0129798868",
    "unlockHex": "<miscData><ancestorTxAVersion><ancestorTxAVin1Header><ancestorTxAVin1MiscData>
    <ancestorTxAVin1DataToCTX><ancestorTxAVin1CTXToScriptCode><ancestorTxAVin1CTXScriptCodeData>
    <ancestorTxAVin1CTXEndNSequence><ancestorTxAVin2Header><ancestorTxAVin2MiscData>
    <ancestorTxAVin2DataToCTX><ancestorTxAVin2CTXToScriptCode><ancestorTxAVin2CTXScriptCodeData>
    <ancestorTxAVin2CTXEndNSequence><ancestorTxAVinFundVin><ancestorTxAVout1Data><ancestorTxAVout2D
↪   ata>
    <ancestorTxABoltPubKeyHash1><ancestorTxABoltPubKeyHash2><ancestorTxAChangeNLockTime><ancestorTx
↪   BVersion>
    <ancestorTxBVin1Header><ancestorTxBVin1MiscData><ancestorTxBVin1DataToCTX><ancestorTxBVin1CTXTo
↪   ScriptCode>
    <ancestorTxBVin1CTXScriptCodeData><ancestorTxBVin1CTXEndNSequence><ancestorTxBVin2Header>
    <ancestorTxBVin2MiscData><ancestorTxBVin2DataToCTX><ancestorTxBVin2CTXToScriptCode>
    <ancestorTxBVin2CTXScriptCodeData><ancestorTxBVin2CTXEndNSequence><ancestorTxBVinFundVin>
    <ancestorTxBVout1Data><ancestorTxBVout2Data><ancestorTxBBoltPubKeyHash1><ancestorTxBBoltPubKeyH
↪   ash2>
    <ancestorTxBChangeNLockTime><grandparentBoltVoutIdx><interopBoltVoutIdx><interopPubKeyHash>
    <interopOutpoint><interopParentOutpoint><interopGenesisOutpoint><interopIssuerPubKey><fundOutpo
↪   int>
    <changeOutput><pubKeyHash1><pubKeyHash2><nextBalanceCommit><nextTxoType><inputIndexN><sig><pubK
↪   ey>
    <ctx>",
    "lockAsm": "<mintData> <balance> <balanceCommit> <pubKeyHash> <pubKeyHashCommit>
↪   <pubKeyHashCommit2>
     <otherParentOutpoint> <otherGrandparentOutpoint> <otherGenesisOutpoint> <otherIssuerPubKey>
↪   <txoType>
     <outputIndexN> <parentOutpoint> <grandparentOutpoint> <genesisOutpoint> <issuerPubKey> OP_SWAP
     OP_BIN2NUM OP_NOT OP_TUCK OP_IF 11 OP_PICK OP_EQUALVERIFY 13 OP_PICK OP_DUP 25 OP_EQUAL OP_NOT
     OP_VERIFY 27 OP_EQUAL OP_NOT OP_VERIFY OP_ELSE OP_DROP OP_ENDIF OP_SWAP OP_DUP OP_BIN2NUM
↪   OP_ONOTEQUAL
     OP_16 OP_PICK OP_IFDUP OP_IF OP_DUP 41 OP_2DROP OP_4 OP_SPLIT 20 OP_SPLIT 20 OP_SPLIT 24
↪   OP_SPLIT OP_1
```

```
 OP_SPLIT OP_SWAP OP_DUP fd OP_EQUAL OP_IF OP_DROP OP_2 OP_SPLIT OP_SWAP OP_ELSE 00 OP_CAT
↪ OP_ENDIF
 OP_BIN2NUM OP_SPLIT OP_8 OP_SPLIT OP_4 OP_SPLIT 20 OP_SPLIT OP_4 OP_SPLIT OP_5 OP_ROLL 19
↪ OP_PICK
 OP_SIZE OP_NIP OP_DUP 4b OP_LESSTHANOREQUAL OP_IF OP_1 OP_ELSE OP_DUP 4c OP_LESSTHANOREQUAL
↪ OP_IF
 OP_2 OP_ELSE OP_DUP 4d OP_LESSTHANOREQUAL OP_IF OP_3 OP_ELSE OP_DUP 4e OP_EQUAL OP_IF OP_5
↪ OP_ENDIF
 OP_ENDIF OP_ENDIF OP_ENDIF OP_ADD OP_SPLIT OP_9 OP_SPLIT OP_9 OP_SPLIT 15 OP_SPLIT 15 OP_SPLIT
↪ 15
 OP_SPLIT 25 OP_SPLIT 25 OP_SPLIT 25 OP_SPLIT 22 OP_SPLIT OP_2 OP_SPLIT OP_1 OP_SPLIT 25
↪ OP_SPLIT 25
 OP_SPLIT 25 OP_SPLIT 22 OP_SPLIT 1f OP_PICK OP_2 OP_MOD OP_DUP OP_IF 2f OP_PICK OP_1ADD 21
↪ OP_PICK
 OP_EQUALVERIFY OP_ENDIF OP_DUP 30 OP_PICK OP_2 OP_MOD OP_SWAP OP_IF 0 OP_ELSE OP_1 OP_ENDIF
↪ OP_EQUALVERIFY
 1d02b0178876a914 ffffffffffff1f00 080000000000000000 1400000000000000000000000000000000000000000
 21000000000000000000000000000000000000000000000000000000000000000000
 240000000000000000000000000000000000000000000000000000000000000000000000 2f OP_PICK OP_DUP 0
 OP_GREATERTHANOREQUAL OP_VERIFY OP_5 OP_PICK OP_LESSTHANOREQUAL OP_VERIFY 36 OP_PICK OP_DUP 0
 OP_GREATERTHANOREQUAL OP_VERIFY OP_5 OP_PICK OP_LESSTHANOREQUAL OP_VERIFY 35 OP_PICK OP_DUP 20
 OP_EQUAL OP_SWAP OP_DUP 21 OP_EQUAL OP_SWAP OP_DUP 22 OP_EQUAL OP_SWAP OP_DUP 23 OP_EQUAL
↪ OP_SWAP
 OP_DUP 24 OP_EQUAL OP_SWAP OP_DUP 25 OP_EQUAL OP_SWAP OP_DUP 26 OP_EQUAL OP_SWAP 27 OP_EQUAL 24
 OP_PICK OP_BIN2NUM OP_1 OP_EQUALVERIFY OP_2 OP_PICK OP_IF 37 OP_PICK 3f OP_PICK OP_ADD OP_13
 OP_PICK OP_LESSTHANOREQUAL OP_VERIFY OP_ENDIF OP_4 OP_PICK OP_IF 37 OP_PICK 3f OP_PICK OP_2DUP
 OP_SUB 0 OP_GREATERTHANOREQUAL OP_VERIFY OP_GREATERTHANOREQUAL OP_VERIFY OP_ENDIF OP_14 OP_PICK
 OP_NOTIF 24 OP_PICK OP_14 OP_PICK OP_CAT OP_DUP 42 OP_PICK OP_CAT 88ac OP_CAT OP_6 OP_PICK OP_3
 OP_PICK OP_BOOLOR OP_IF OP_SWAP 41 OP_PICK OP_CAT 88ac OP_CAT 3e OP_PICK OP_IF OP_SWAP OP_ENDIF
 OP_CAT OP_ELSE OP_NIP OP_ENDIF OP_ELSE ff OP_ENDIF 02b017 OP_8 OP_PICK OP_10 OP_PICK OP_BOOLOR
 OP_4 OP_PICK OP_BOOLOR OP_IF 20 OP_PICK OP_CAT OP_13 OP_PICK OP_ELSE OP_2 OP_PICK OP_5 OP_PICK
 OP_BOOLOR OP_7 OP_PICK OP_BOOLOR OP_IF 20 OP_PICK 08 42 OP_PICK OP_CAT OP_6 OP_PICK 41 OP_PICK
 OP_BOOLAND OP_IF OP_SWAP OP_ENDIF OP_CAT OP_ELSE OP_5 OP_PICK OP_IF 08 3a OP_PICK 3a OP_PICK
↪ OP_ADD
 OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT
↪ OP_ENDIF
 OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT
↪ OP_ENDIF
 OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT
↪ OP_ENDIF
 OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT
↪ OP_ENDIF
 OP_CAT OP_14 OP_PICK OP_CAT OP_ELSE OP_7 OP_PICK OP_IF 08 3a OP_PICK 3a OP_PICK OP_SUB OP_SIZE
↪ OP_8
 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE
↪ OP_8
 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE
↪ OP_8
 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE
↪ OP_8
 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_CAT OP_ENDIF OP_CAT
↪ OP_14
 OP_PICK OP_CAT OP_ENDIF OP_ENDIF OP_ENDIF OP_ENDIF OP_CAT OP_8 OP_PICK OP_IF 1e OP_PICK OP_CAT
↪ 14 43
 OP_PICK OP_CAT OP_CAT OP_12 OP_PICK OP_ELSE OP_9 OP_PICK OP_6 OP_PICK OP_BOOLOR OP_8 OP_PICK
↪ OP_BOOLOR
 OP_IF 1d OP_PICK OP_CAT OP_12 OP_PICK OP_DUP OP_CAT OP_ELSE OP_2 OP_PICK OP_7 OP_PICK
↪ OP_BOOLOR OP_IF
```

```
    1e OP_PICK OP_CAT 14 43 OP_PICK OP_CAT 14 43 OP_PICK OP_CAT OP_CAT OP_ELSE OP_3 OP_PICK OP_IF
↪ 1c OP_PICK
    OP_CAT OP_12 OP_PICK OP_DUP OP_CAT OP_ELSE OP_4 OP_PICK OP_IF 3e OP_PICK OP_NOTIF 1e OP_PICK
↪ OP_ELSE 14
    4a OP_PICK OP_CAT OP_ENDIF OP_CAT 14 43 OP_PICK OP_CAT OP_CAT OP_12 OP_PICK OP_ENDIF OP_ENDIF
↪ OP_ENDIF
    OP_ENDIF OP_ENDIF OP_CAT OP_8 OP_PICK OP_10 OP_PICK OP_BOOLOR OP_4 OP_PICK OP_BOOLOR OP_6
↪ OP_PICK OP_BOOLOR
    OP_7 OP_PICK OP_BOOLOR OP_8 OP_PICK OP_BOOLOR OP_IF OP_10 OP_PICK OP_CAT OP_10 OP_PICK OP_CAT
↪ OP_10 OP_PICK
    OP_CAT OP_11 OP_PICK OP_CAT OP_ELSE OP_2 OP_PICK OP_IF 24 49 OP_PICK OP_CAT OP_CAT 24 48
↪ OP_PICK OP_CAT OP_CAT
    24 47 OP_PICK OP_CAT OP_CAT 21 46 OP_PICK OP_CAT OP_CAT OP_ELSE OP_4 OP_PICK OP_IF 3e OP_PICK
↪ OP_NOTIF 24 49
    OP_PICK OP_CAT OP_CAT 24 48 OP_PICK OP_CAT OP_CAT OP_ELSE 24 28 OP_PICK OP_CAT OP_CAT 15
↪ OP_PICK OP_CAT OP_ENDIF
    OP_10 OP_PICK OP_CAT OP_11 OP_PICK OP_CAT OP_ENDIF OP_ENDIF OP_ENDIF 01 40 OP_PICK OP_CAT
↪ OP_CAT OP_4 OP_PICK
    3f OP_PICK OP_BOOLAND OP_IF 00 OP_CAT 24 49 OP_PICK OP_CAT OP_CAT 24 48 OP_PICK OP_CAT OP_CAT
↪ 24 47 OP_PICK
    OP_CAT OP_CAT OP_ELSE OP_3 OP_PICK OP_IF 2f OP_ELSE 3e OP_ENDIF OP_PICK OP_NOTIF 00 OP_ELSE 51
↪ OP_ENDIF OP_CAT
    24 28 OP_PICK OP_CAT OP_CAT 15 OP_PICK OP_CAT 2d OP_PICK OP_IF 24 28 OP_PICK OP_CAT OP_ELSE 13
↪ OP_PICK OP_ENDIF
    OP_CAT OP_ENDIF 12 OP_PICK OP_CAT 11 OP_PICK OP_CAT OP_SIZE fd OP_SWAP OP_CAT OP_TUCK 28
↪ OP_PICK OP_SWAP OP_CAT
    OP_SWAP OP_CAT OP_3 OP_PICK OP_5 OP_PICK OP_BOOLOR OP_9 OP_PICK OP_BOOLOR OP_IF 02b017 OP_4
↪ OP_PICK OP_IF 08 43
    OP_PICK OP_CAT 23 OP_PICK OP_CAT OP_CAT 14 4c OP_PICK OP_CAT OP_CAT 14 44 OP_PICK OP_CAT 14 46
↪ OP_PICK OP_CAT
    OP_CAT OP_CAT 24 2a OP_PICK OP_CAT OP_CAT 17 OP_PICK OP_CAT 15 OP_PICK OP_CAT 14 OP_PICK
↪ OP_CAT OP_ELSE
      21 OP_PICK OP_CAT OP_15 OP_PICK OP_CAT OP_9 OP_PICK OP_NOTIF 1f OP_ELSE 1e OP_ENDIF OP_PICK
↪ OP_CAT
      OP_14 OP_PICK OP_DUP OP_CAT OP_CAT OP_12 OP_PICK OP_CAT OP_12 OP_PICK OP_CAT OP_12 OP_PICK
↪ OP_CAT
      OP_13 OP_PICK OP_CAT OP_ENDIF 01 42 OP_PICK OP_CAT OP_CAT OP_4 OP_PICK OP_IF 40 OP_ELSE 31
↪ OP_ENDIF
      OP_PICK OP_NOTIF 51 OP_ELSE 00 OP_ENDIF OP_CAT OP_5 OP_PICK OP_IF 24 2a OP_PICK 20 OP_SPLIT
↪ OP_NOT
      OP_IF 01000000 OP_ELSE 00000000 OP_ENDIF OP_CAT OP_DUP OP_TOALTSTACK OP_CAT OP_CAT 1d OP_PICK
↪ OP_CAT
      1b OP_PICK OP_CAT 1a OP_PICK OP_CAT OP_ELSE OP_9 OP_PICK OP_IF 24 2a OP_PICK OP_CAT OP_CAT 17
↪ OP_PICK
      OP_CAT 15 OP_PICK OP_CAT 14 OP_PICK OP_CAT OP_ELSE 24 4b OP_PICK OP_CAT OP_CAT 24 4a OP_PICK
↪ OP_CAT
      OP_CAT 24 49 OP_PICK OP_CAT OP_CAT 21 48 OP_PICK OP_CAT OP_CAT OP_ENDIF OP_ENDIF 13 OP_PICK
↪ OP_CAT
      OP_2 OP_PICK 29 OP_PICK OP_SWAP OP_CAT OP_SWAP OP_CAT OP_4 OP_PICK OP_IF 40 OP_ELSE 31
↪ OP_ENDIF OP_PICK
      OP_IF OP_SWAP OP_ENDIF OP_CAT OP_ENDIF OP_2 OP_ROLL 11 OP_PICK OP_NOTIF OP_CAT OP_ELSE
↪ OP_DROP OP_ENDIF
      43 OP_PICK OP_CAT OP_HASH256 24 OP_PICK OP_EQUALVERIFY 26 OP_PICK OP_2 OP_PICK OP_5 OP_PICK
↪ OP_BOOLOR
      OP_IF 48 OP_PICK 3f OP_PICK OP_IF OP_SWAP OP_ENDIF OP_CAT OP_ELSE OP_3 OP_PICK OP_IF
↪ OP_FROMALTSTACK
      30 OP_PICK OP_IF OP_SWAP OP_ENDIF OP_CAT OP_ENDIF OP_ENDIF OP_16 OP_PICK 2c OP_PICK
↪ OP_BOOLAND OP_IF
```

```
    OP_DUP 01 OP_EQUAL OP_SWAP OP_DUP 03 OP_EQUAL OP_SWAP OP_DUP 05 OP_EQUAL OP_SWAP 07 OP_EQUAL
↪ 71 OP_PICK
    6b OP_PICK OP_SIZE 0 OP_GREATERTHAN OP_NIP OP_DUP OP_IF OP_3 OP_ELSE OP_2 OP_ENDIF OP_2
↪ OP_ROLL OP_SWAP
    OP_CAT 72 OP_PICK OP_CAT 71 OP_PICK OP_SIZE OP_DUP 4b OP_LESSTHANOREQUAL OP_IF OP_ELSE OP_DUP
↪ 4c
    OP_LESSTHANOREQUAL OP_IF 4c OP_SWAP OP_1 OP_ELSE OP_DUP 4d OP_LESSTHANOREQUAL OP_IF 4d
↪ OP_SWAP OP_2
    OP_ELSE OP_DUP 4e OP_SWAP OP_4 OP_ENDIF OP_ENDIF OP_NUM2BIN OP_CAT OP_ENDIF OP_SWAP OP_CAT
↪ OP_CAT
    70 OP_PICK OP_CAT 6f OP_PICK OP_CAT 6e OP_PICK 14 OP_PICK OP_CAT OP_SIZE OP_DUP ffff
↪ OP_BIN2NUM OP_SWAP
    OP_LESSTHANOREQUAL OP_IF fd OP_SWAP OP_ELSE OP_DUP ffffffff OP_BIN2NUM OP_SWAP
↪ OP_LESSTHANOREQUAL OP_IF
    fe OP_SWAP OP_SIZE OP_3 OP_EQUAL OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_ELSE ff OP_SWAP OP_SIZE
↪ OP_8
    OP_LESSTHAN OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_SWAP OP_CAT
↪ OP_ENDIF
    OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_ENDIF OP_ENDIF OP_CAT OP_SWAP
↪ OP_CAT OP_CAT
    6d OP_PICK OP_CAT OP_SWAP OP_IF 6b OP_PICK OP_CAT 6a OP_PICK OP_SIZE OP_DUP 4b
↪ OP_LESSTHANOREQUAL OP_IF
    OP_ELSE OP_DUP 4c OP_LESSTHANOREQUAL OP_IF 4c OP_SWAP OP_1 OP_ELSE OP_DUP 4d
↪ OP_LESSTHANOREQUAL OP_IF
    4d OP_SWAP OP_2 OP_ELSE OP_DUP 4e OP_SWAP OP_4 OP_ENDIF OP_ENDIF OP_NUM2BIN OP_CAT OP_ENDIF
↪ OP_SWAP
    OP_CAT OP_CAT 69 OP_PICK OP_CAT 68 OP_PICK OP_CAT 67 OP_PICK 13 OP_PICK OP_CAT OP_SIZE OP_DUP
↪ ffff
    OP_BIN2NUM OP_SWAP OP_LESSTHANOREQUAL OP_IF fd OP_SWAP OP_ELSE OP_DUP ffffffff OP_BIN2NUM
↪ OP_SWAP
    OP_LESSTHANOREQUAL OP_IF fe OP_SWAP OP_SIZE OP_3 OP_EQUAL OP_IF 00 OP_SWAP OP_CAT OP_ENDIF
↪ OP_ELSE ff
    OP_SWAP OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN
↪ OP_IF 00
    OP_SWAP OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_ENDIF
↪ OP_ENDIF
    OP_CAT OP_SWAP OP_CAT OP_CAT 66 OP_PICK OP_CAT OP_ENDIF 65 OP_PICK OP_CAT 63 OP_PICK OP_SIZE 0
    OP_GREATERTHAN OP_NIP 62 OP_PICK OP_SIZE 0 OP_GREATERTHAN OP_NIP OP_2DUP OP_BOOLAND OP_IF
↪ OP_5 OP_ELSE
    OP_2DUP OP_BOOLOR OP_IF OP_4 OP_ELSE OP_3 OP_ENDIF OP_ENDIF OP_3 OP_ROLL OP_SWAP OP_CAT 29
↪ OP_PICK
    OP_CAT OP_4 OP_PICK OP_CAT 66 OP_PICK OP_CAT 14 OP_PICK OP_CAT OP_2 OP_ROLL OP_IF 28 OP_PICK
↪ OP_CAT
    OP_3 OP_PICK OP_CAT 64 OP_PICK OP_CAT 13 OP_PICK OP_CAT OP_ENDIF 28 OP_PICK OP_CAT 11 OP_PICK
↪ OP_CAT
    63 OP_PICK OP_CAT 88ac OP_CAT OP_SWAP OP_IF 27 OP_PICK OP_CAT OP_16 OP_PICK OP_CAT 61 OP_PICK
↪ OP_CAT
    88ac OP_CAT OP_ENDIF 60 OP_PICK OP_CAT OP_HASH256 OP_DUP 2e OP_PICK 20 OP_SPLIT OP_DROP
↪ OP_EQUALVERIFY
    4c OP_PICK OP_CAT OP_6 OP_PICK OP_5 OP_PICK OP_BOOLOR OP_IF 5f OP_PICK 59 OP_PICK OP_SIZE 0
    OP_GREATERTHAN OP_NIP OP_DUP OP_IF OP_3 OP_ELSE OP_2 OP_ENDIF OP_2 OP_ROLL OP_SWAP OP_CAT 60
↪ OP_PICK
    OP_CAT 5f OP_PICK OP_SIZE OP_DUP 4b OP_LESSTHANOREQUAL OP_IF OP_ELSE OP_DUP 4c
↪ OP_LESSTHANOREQUAL OP_IF
    4c OP_SWAP OP_1 OP_ELSE OP_DUP 4d OP_LESSTHANOREQUAL OP_IF 4d OP_SWAP OP_2 OP_ELSE OP_DUP 4e
↪ OP_SWAP
    OP_4 OP_ENDIF OP_ENDIF OP_NUM2BIN OP_CAT OP_ENDIF OP_SWAP OP_CAT OP_CAT 5e OP_PICK OP_CAT 5d
↪ OP_PICK
```

```
      OP_CAT 5c OP_PICK 15 OP_PICK OP_CAT OP_SIZE OP_DUP ffff OP_BIN2NUM OP_SWAP OP_LESSTHANOREQUAL
↪  OP_IF fd
      OP_SWAP OP_ELSE OP_DUP ffffffff OP_BIN2NUM OP_SWAP OP_LESSTHANOREQUAL OP_IF fe OP_SWAP
↪  OP_SIZE OP_3
      OP_EQUAL OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_ELSE ff OP_SWAP OP_SIZE OP_8 OP_LESSTHAN OP_IF
↪  00 OP_SWAP
      OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_SIZE OP_8
↪  OP_LESSTHAN
      OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_ENDIF OP_ENDIF OP_CAT OP_SWAP OP_CAT OP_CAT 5b OP_PICK
↪  OP_CAT
      OP_SWAP OP_IF 59 OP_PICK OP_CAT 58 OP_PICK OP_SIZE OP_DUP 4b OP_LESSTHANOREQUAL OP_IF OP_ELSE
↪  OP_DUP 4c
      OP_LESSTHANOREQUAL OP_IF 4c OP_SWAP OP_1 OP_ELSE OP_DUP 4d OP_LESSTHANOREQUAL OP_IF 4d
↪  OP_SWAP OP_2
      OP_ELSE OP_DUP 4e OP_SWAP OP_4 OP_ENDIF OP_ENDIF OP_NUM2BIN OP_CAT OP_ENDIF OP_SWAP OP_CAT
↪  OP_CAT 57
      OP_PICK OP_CAT 56 OP_PICK OP_CAT 55 OP_PICK 13 OP_PICK OP_CAT OP_SIZE OP_DUP ffff OP_BIN2NUM
↪  OP_SWAP
      OP_LESSTHANOREQUAL OP_IF fd OP_SWAP OP_ELSE OP_DUP ffffffff OP_BIN2NUM OP_SWAP
↪  OP_LESSTHANOREQUAL OP_IF
      fe OP_SWAP OP_SIZE OP_3 OP_EQUAL OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_ELSE ff OP_SWAP OP_SIZE
↪  OP_8
      OP_LESSTHAN OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_SWAP OP_CAT
↪  OP_ENDIF
      OP_SIZE OP_8 OP_LESSTHAN OP_IF 00 OP_SWAP OP_CAT OP_ENDIF OP_ENDIF OP_ENDIF OP_CAT OP_SWAP
↪  OP_CAT OP_CAT
      54 OP_PICK OP_CAT OP_ENDIF 53 OP_PICK OP_CAT 51 OP_PICK OP_SIZE 0 OP_GREATERTHAN OP_NIP 50
↪  OP_PICK
      OP_SIZE 0 OP_GREATERTHAN OP_NIP OP_2DUP OP_BOOLAND OP_IF OP_5 OP_ELSE OP_2DUP OP_BOOLOR OP_IF
↪  OP_4
      OP_ELSE OP_3 OP_ENDIF OP_ENDIF OP_3 OP_ROLL OP_SWAP OP_CAT 2a OP_PICK OP_CAT
      OP_5 OP_PICK OP_CAT 54 OP_PICK OP_CAT 15 OP_PICK OP_CAT OP_2 OP_ROLL OP_IF 29 OP_PICK OP_CAT
↪  OP_4
      OP_PICK OP_CAT 52 OP_PICK OP_CAT 14 OP_PICK OP_CAT OP_ENDIF 29 OP_PICK OP_CAT 12 OP_PICK
↪  OP_CAT 51
      OP_PICK OP_CAT 88ac OP_CAT OP_SWAP OP_IF 28 OP_PICK OP_CAT 11 OP_PICK OP_CAT 4f OP_PICK
↪  OP_CAT 88ac
      OP_CAT OP_ENDIF 4e OP_PICK OP_CAT OP_HASH256 OP_DUP 36 OP_PICK 20 OP_SPLIT OP_DROP
↪  OP_EQUALVERIFY 4c
      OP_PICK OP_CAT 31 OP_PICK OP_IF OP_SWAP OP_ENDIF OP_CAT OP_ENDIF OP_CAT OP_ELSE 71 OP_PICK 71
↪  OP_PICK
      71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71
↪  OP_PICK 71
      OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71
↪  OP_PICK 71
      OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71
↪  OP_PICK 71
      OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71 OP_PICK 71
↪  OP_PICK OP_CAT
      OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT
↪  OP_CAT
      OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT
↪  OP_CAT
      OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_CAT OP_SIZE OP_NIP 0 OP_EQUALVERIFY
↪  OP_ENDIF 44
      OP_PICK OP_CAT OP_HASH256 29 OP_PICK OP_EQUALVERIFY OP_ENDIF",
    "unlockAsm": "<miscData> <ancestorTxAVersion> <ancestorTxAVin1Header> <ancestorTxAVin1MiscData>
↪  <ancestorTxAVin1DataToCTX>
```

```
    <ancestorTxAVin1CTXToScriptCode> <ancestorTxAVin1CTXScriptCodeData>
↪   <ancestorTxAVin1CTXEndNSequence>
    <ancestorTxAVin2Header> <ancestorTxAVin2MiscData> <ancestorTxAVin2DataToCTX>
↪   <ancestorTxAVin2CTXToScriptCode>
    <ancestorTxAVin2CTXScriptCodeData> <ancestorTxAVin2CTXEndNSequence> <ancestorTxAVinFundVin>
↪   <ancestorTxAVout1Data>
    <ancestorTxAVout2Data> <ancestorTxABoltPubKeyHash1> <ancestorTxABoltPubKeyHash2>
↪   <ancestorTxAChangeNLockTime>
    <ancestorTxBVersion> <ancestorTxBVin1Header> <ancestorTxBVin1MiscData> <ancestorTxBVin1DataToCT
X>
    <ancestorTxBVin1CTXToScriptCode> <ancestorTxBVin1CTXScriptCodeData>
↪   <ancestorTxBVin1CTXEndNSequence>
    <ancestorTxBVin2Header> <ancestorTxBVin2MiscData> <ancestorTxBVin2DataToCTX>
↪   <ancestorTxBVin2CTXToScriptCode>
    <ancestorTxBVin2CTXScriptCodeData> <ancestorTxBVin2CTXEndNSequence> <ancestorTxBVinFundVin>
↪   <ancestorTxBVout1Data>
    <ancestorTxBVout2Data> <ancestorTxBBoltPubKeyHash1> <ancestorTxBBoltPubKeyHash2>
↪   <ancestorTxBChangeNLockTime>
    <grandparentBoltVoutIdx> <interopBoltVoutIdx> <interopPubKeyHash> <interopOutpoint>
↪   <interopParentOutpoint>
    <interopGenesisOutpoint> <interopIssuerPubKey> <fundOutpoint> <changeOutput> <pubKeyHash1>
↪   <pubKeyHash2>
    <nextBalanceCommit> <nextTxoType> <inputIndexN> <sig> <pubKey> <ctx>",
    "lockArgs": [
      "mintData",
      "balance",
      "balanceCommit",
      "pubKeyHash",
      "pubKeyHashCommit",
      "pubKeyHashCommit2",
      "otherParentOutpoint",
      "otherGrandparentOutpoint",
      "otherGenesisOutpoint",
      "otherIssuerPubKey",
      "txoType",
      "outputIndexN",
      "parentOutpoint",
      "grandparentOutpoint",
      "genesisOutpoint",
      "issuerPubKey"
    ],
    "unlockArgs": [
      "miscData",
      "ancestorTxAVersion",
      "ancestorTxAVin1Header",
      "ancestorTxAVin1MiscData",
      "ancestorTxAVin1DataToCTX",
      "ancestorTxAVin1CTXToScriptCode",
      "ancestorTxAVin1CTXScriptCodeData",
      "ancestorTxAVin1CTXEndNSequence",
      "ancestorTxAVin2Header",
      "ancestorTxAVin2MiscData",
      "ancestorTxAVin2DataToCTX",
      "ancestorTxAVin2CTXToScriptCode",
      "ancestorTxAVin2CTXScriptCodeData",
      "ancestorTxAVin2CTXEndNSequence",
      "ancestorTxAVinFundVin",
      "ancestorTxAVout1Data",
```

```
      "ancestorTxAVout2Data",
      "ancestorTxABoltPubKeyHash1",
      "ancestorTxABoltPubKeyHash2",
      "ancestorTxAChangeNLockTime",
      "ancestorTxBVersion",
      "ancestorTxBVin1Header",
      "ancestorTxBVin1MiscData",
      "ancestorTxBVin1DataToCTX",
      "ancestorTxBVin1CTXToScriptCode",
      "ancestorTxBVin1CTXScriptCodeData",
      "ancestorTxBVin1CTXEndNSequence",
      "ancestorTxBVin2Header",
      "ancestorTxBVin2MiscData",
      "ancestorTxBVin2DataToCTX",
      "ancestorTxBVin2CTXToScriptCode",
      "ancestorTxBVin2CTXScriptCodeData",
      "ancestorTxBVin2CTXEndNSequence",
      "ancestorTxBVinFundVin",
      "ancestorTxBVout1Data",
      "ancestorTxBVout2Data",
      "ancestorTxBBoltPubKeyHash1",
      "ancestorTxBBoltPubKeyHash2",
      "ancestorTxBChangeNLockTime",
      "grandparentBoltVoutIdx",
      "interopBoltVoutIdx",
      "interopPubKeyHash",
      "interopOutpoint",
      "interopParentOutpoint",
      "interopGenesisOutpoint",
      "interopIssuerPubKey",
      "fundOutpoint",
      "changeOutput",
      "pubKeyHash1",
      "pubKeyHash2",
      "nextBalanceCommit",
      "nextTxoType",
      "inputIndexN",
      "sig",
      "pubKey",
      "ctx"
    ],
  }
}
```

References

[1] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.

[2] Scrypt-Inc. *The Trillion Dollar Back-to-the-Genesis Problem*. Accessed: 10 June 2024. 2024. URL: https://archive.is/vVp3T.

[3] ftrader. *SigHashPreimage*. Markup. URL: https://github.com/bitcoin-sv/bitcoin-sv/blob/master/doc/abc/replay-protected-sighash.md#digest-algorithm.

[4] Bitcoin Association. *SIGHASH Flags*. Updated: 21 April 2022, at 05:42. 2022. URL: https://wiki.bitcoinsv.io/index.php/SIGHASH_flags.

[5] nChain. *PUSHTX and its Building Blocks*. White Paper. Updated: 14/12/2021. 2021. URL: https://nchain.com/wp-content/uploads/2022/03/WP1605_PUSHTX-and-its-Building-Blocks.pdf.

[6] Massimo Bartoletti, Stefano Lande, and Roberto Zunino. *Computationally Sound Bitcoin Tokens*. Università degli Studi di Cagliari; Università degli Studi di Trento. bart@unica.it; lande@unica.it; roberto.zunino@unitn.it. Cagliari, Italy; Trento, Italy, 2021.

[7] Satoshi Nakamoto. *Re: Transactions and Scripts*. Accessed: 2024-06-21. 2010. URL: https://bitcointalk.org/index.php?topic=195.msg1611.

[8] ChatGPT. *Crypto Market Cap Inquiry*. Accessed: 12 June 2024. 2024. URL: https://chatgpt.com/share/6ce06f70-f618-48f1-b287-9c63584a6842.

[9] BBC. *CryptoKitties craze slows down transactions on Ethereum*. Accessed: 10-December-2024. 2017. URL: https://archive.is/1AhJb.

[10] Satoshi Nakamoto. *Disable opcodes with potential for exponential resource use*. 2010. URL: https://github.com/bitcoin/bitcoin/commit/4bd188c4383d6e614e18f79dc337fbabe8464c82.

[11] Satoshi. *Bitcoin was created for everyone*. Accessed: 25-Oct-2023. 2023. URL: https://archive.is/sfTrW.

[12] Tucker Carlson. *'They can get you if they want you ' — Roger Ver breaks silence after arrest with Tucker Carlson, talks corruption, privacy coins, and BTC's demise*. Accessed: 10-Dec-2024. 2024. URL: https://archive.is/1AhJb.

[13] *Cardano founder Charles Hoskinson: Founders must decouple from the cryptocurrency*. Accessed: 10-Dec-2024. 2020. URL: https://archive.is/AxfSV.

[14] Insha Zia. *Cardano: Hoskinson Rescinds Web Summit Appearance*. Accessed: 10-Dec-2024. 2023. URL: https://archive.is/nwzWq.

[15] Craig S. Wright. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Tech. rep. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[16] *Cardano price today, ADA to USD live price, marketcap and chart*. Accessed: 10-Dec-2024. 2024. URL: https://archive.is/5GgSf.