BOLT: A Bitcoin Transaction Latching Mechanism & Token Protocol

Frederick Liam Simon Honohan

February 11, 2025

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 [1] 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

David C. Wyld et al. (Eds): ADCO, CMCA, NeCoM, NLPI, CSITEC, BDBS, DMML, SP – 2025
pp. 197-215, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.150516

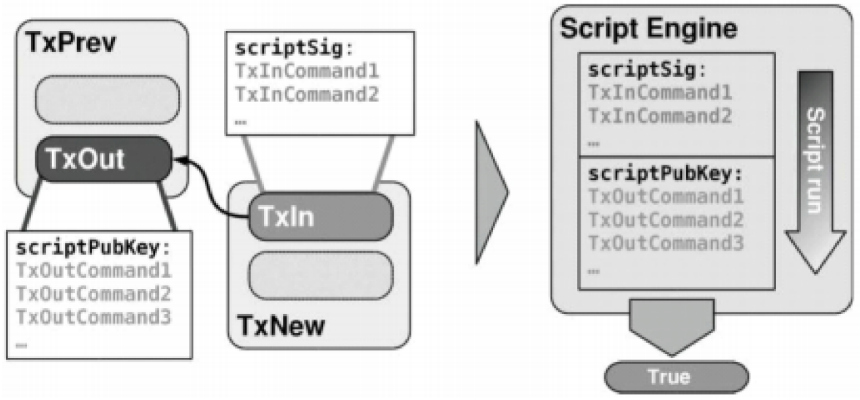

Figure 1: 'TxNew' referencing 'TxPrev' & the order in which the interface scripts are executed.

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* [2]. 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 [3] 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'* [4] 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

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

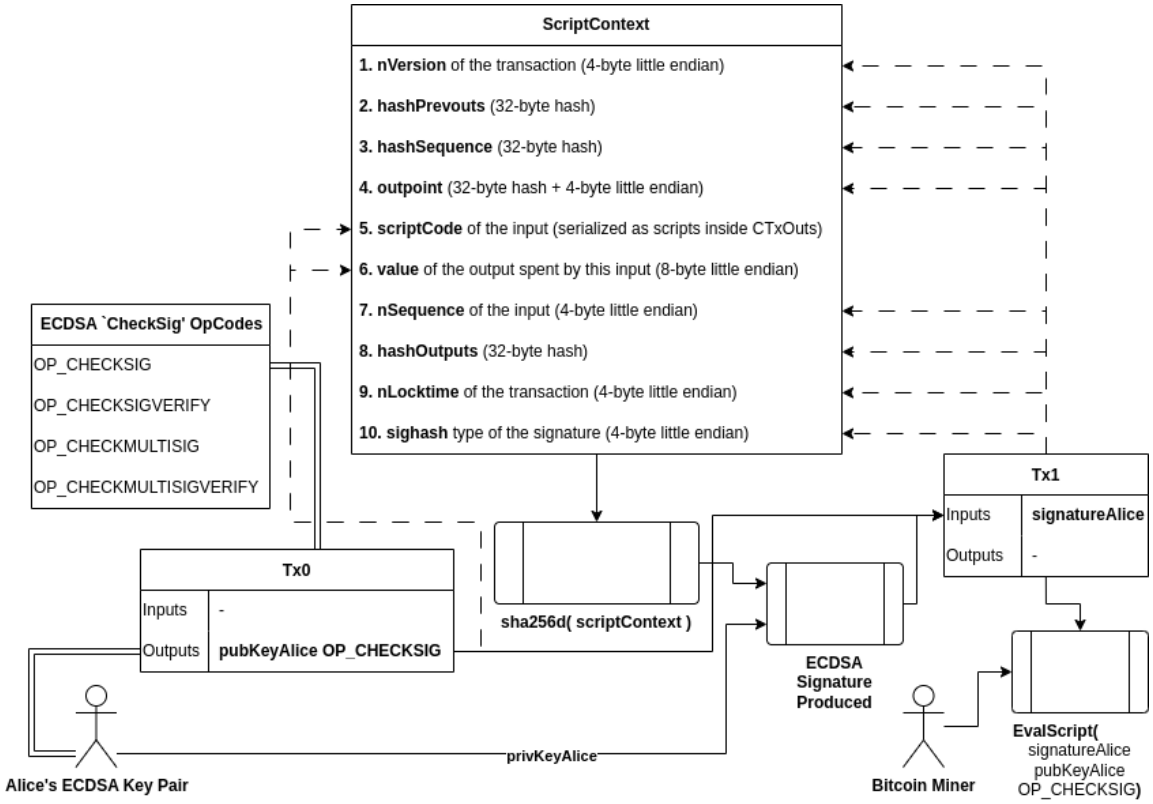

Figure 2: The ScriptContext data structure & ECDSA Signature process governing its resultant hash value.

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.

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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.

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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.

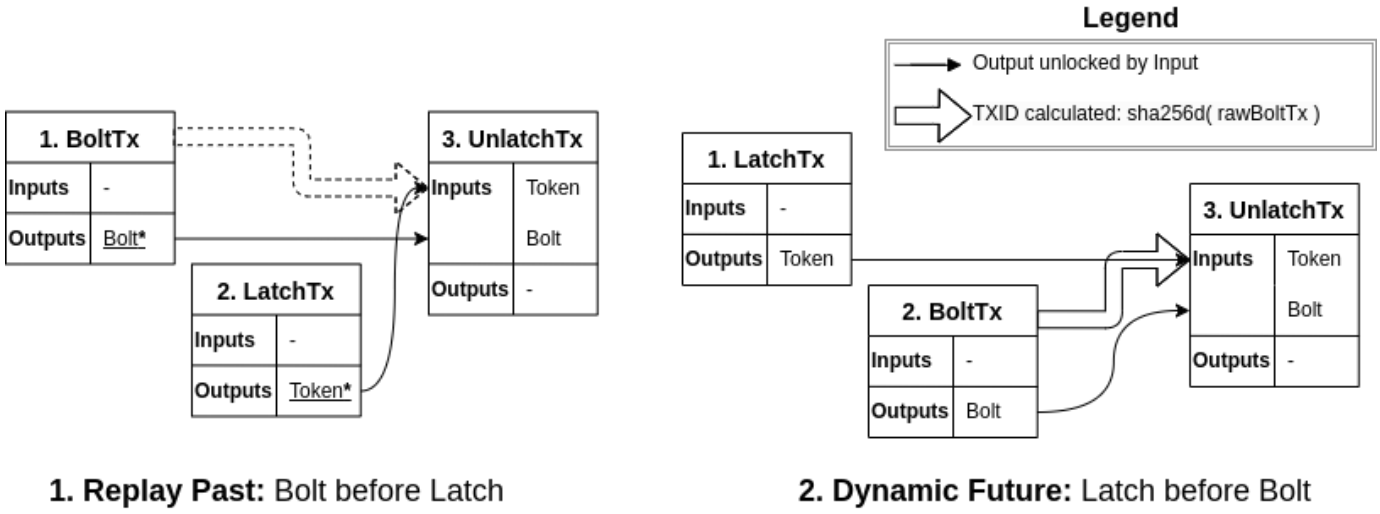

Figure 3: Different time based configurations of disconnected Bolts ("MicroServices").

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.

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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:

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

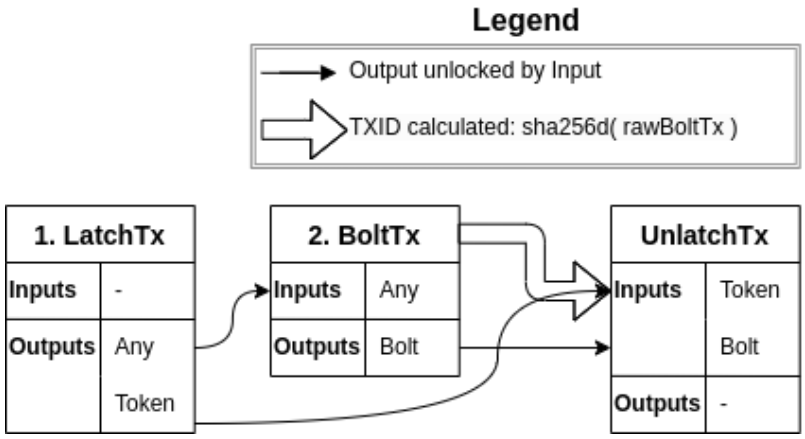

Figure 4: Control transfer by related descendant e.g. Bolt from Future!

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.

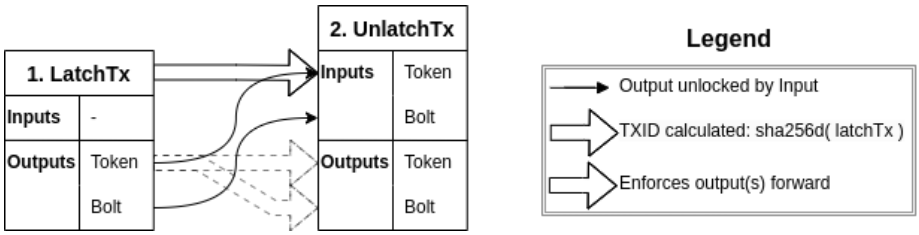

Figure 5: Recovery latch mechanism, with the Token enforcing both the Bolt &the Token itself forward.

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

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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.

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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.

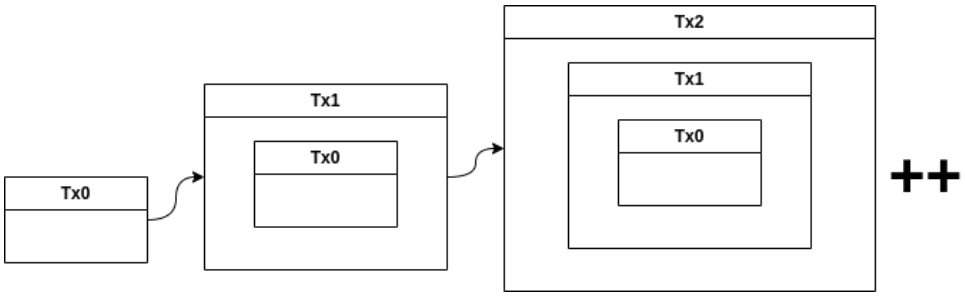

Figure 6: Transaction history bloat problem

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 [5]: *"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.

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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.' [6]*

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.' [7].

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).

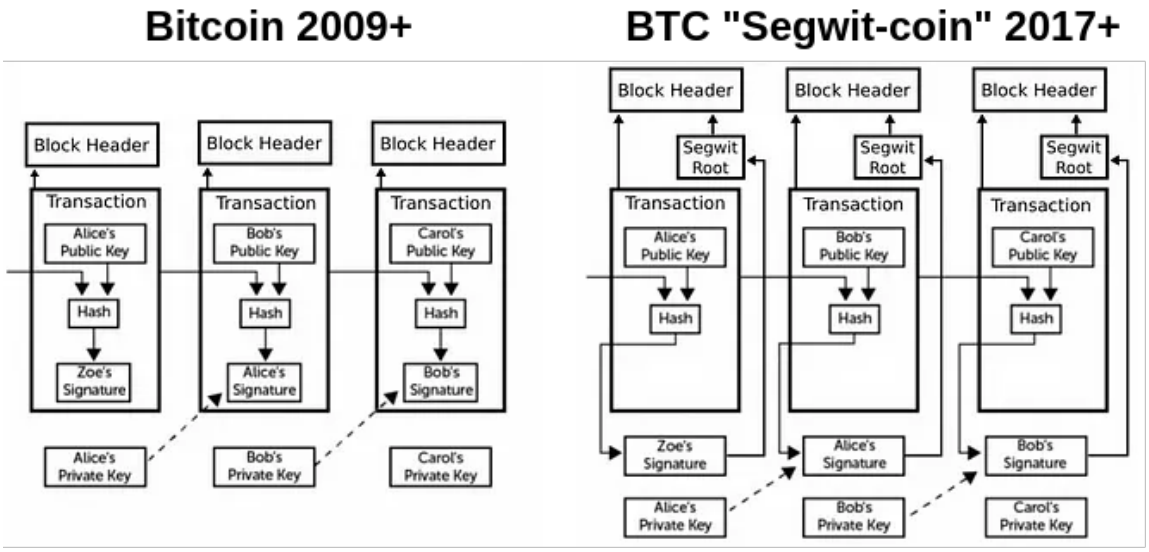

Figure 7: Block & transaction structure differences introduced by SegWit.

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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*

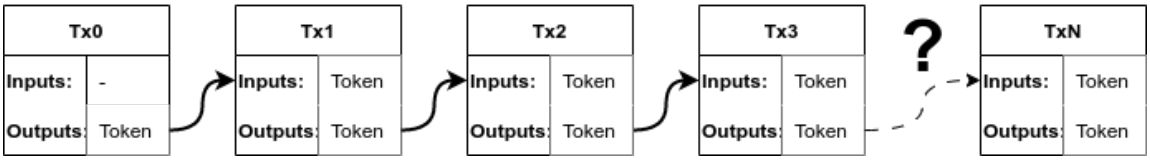

Figure 8: The *Back-To-Genesis Problem*

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

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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.

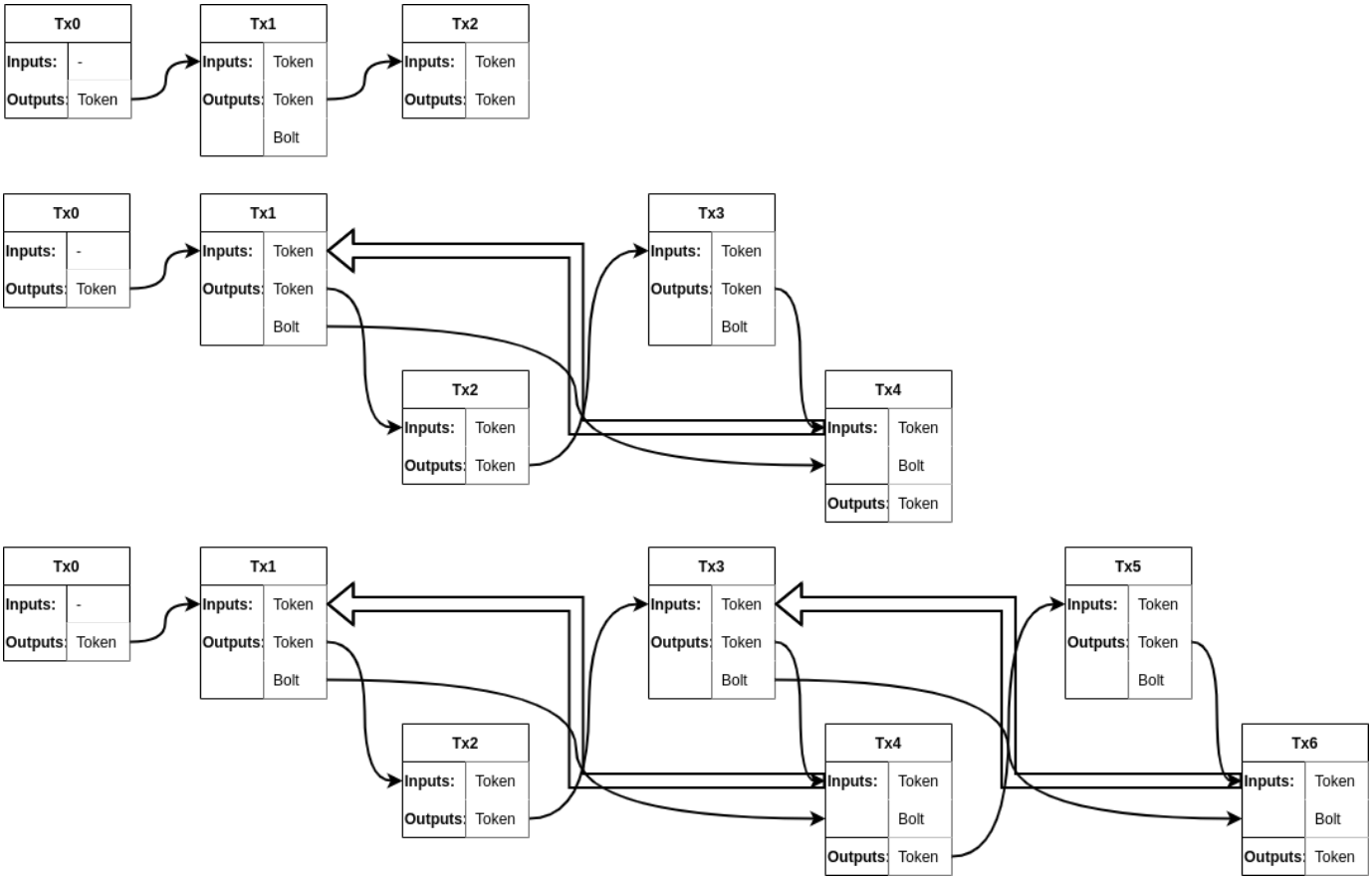

Figure 9: A BOLT-based back-to-genesis token protocol solution

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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.

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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.

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

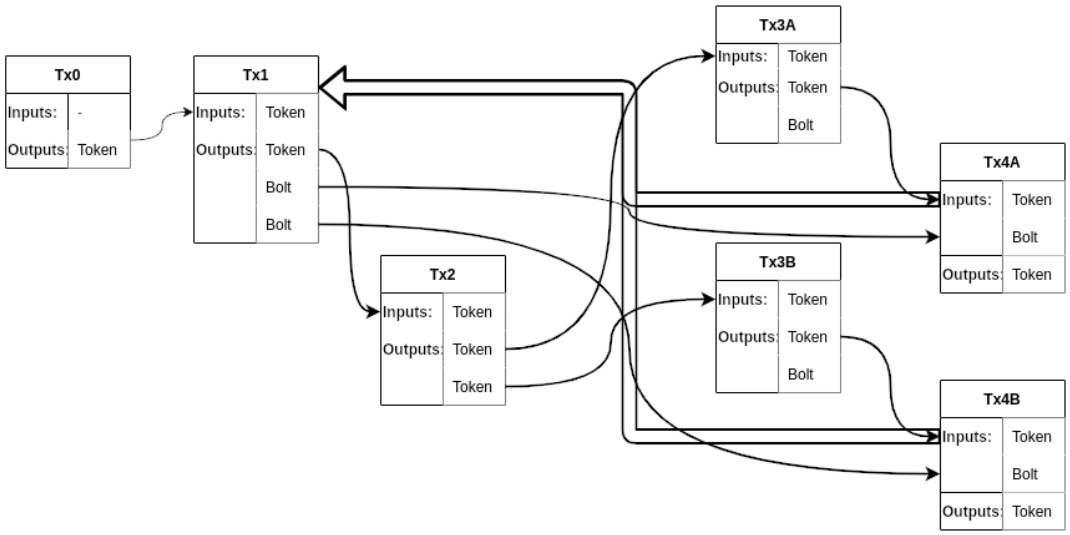

Figure 10: BOLT Protocol Fungible Split Example

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 [8] & cost of transaction today).

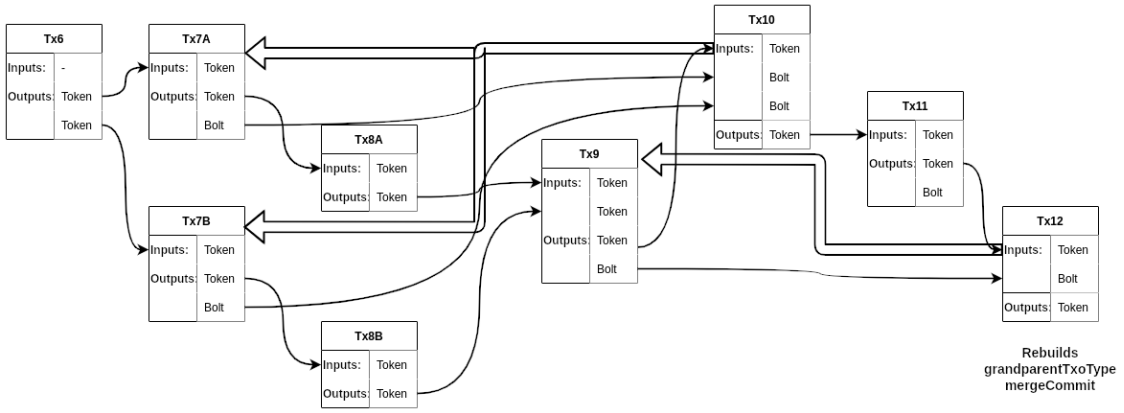

Figure 11: BOLT Protocol Fungible Merge Example

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.

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

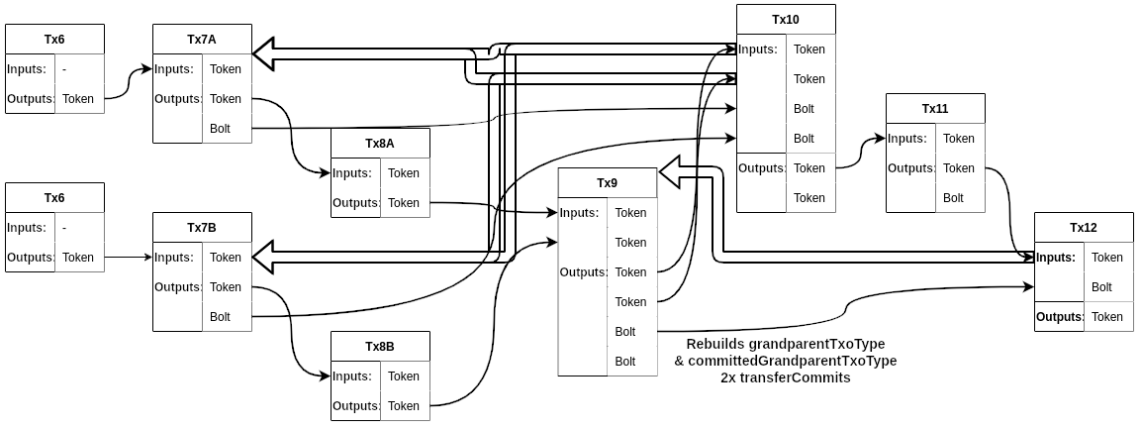

Figure 12: BOLT Protocol Fungible Swap Example

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 [9]. 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

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

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'* [10]. 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 [11], 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.

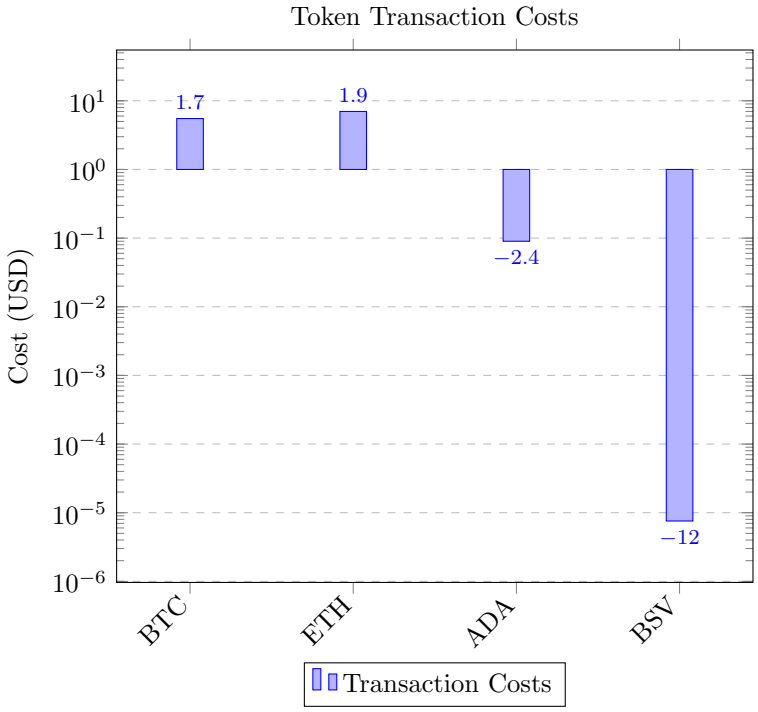

Figure 13: Token Transaction Costs Comparison (log scale). BTC: Bitcoin Ordinals, ETH: ERC20 tokens, ADA: Cardano tokens, BSV: BSV tokens

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

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

[12], [13], an Ethereum founding developer clamouring for control of this emerging technology), than the original Bitcoin (SV) developed by Australian Computer Scientist Craig S. Wright [14]! 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 [15]!

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 [14]) 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**

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

References

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

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

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

[4] 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.

[5] 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.

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

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

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

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

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

[11] 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.

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

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

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

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

UK patent application GB2318902.0 (pending) – available on request. Copyright www.elas.co © 2024

© 2025 By AIRCC Publishing Corporation. This article is published under the Creative Common Attribution (CC BY) license.