Tokeda Criticism

Tokeda is being heavily promoted to the public as an alternative to Group tokenization, so I would like to offer the following detailed review. For your reference, the Tokeda specification is located at: http://media.lokad.com/bitcoin/tokeda-2018-04-30.pdf. Although this specification marks itself as "Early Draft: Incomplete" it is being presented as a viable technology so needs to be analyzed now.

Blockchain and Network Utilization

One of the primary criticisms of on-chain tokenization is that it consumes precious blockchain and UTXO space so I'll lead off with that. Let's see how Tokeda compares.

"Alice signals a token transfer intent to Trent pointing out Bob as the recipient. This requires one Bitcoin transaction, the signaling transaction.
Trent settles the token transfer by indicating that Bob is the new token owner. This requires a second Bitcoin transaction, the settlement transaction." (page 4)

Since token transfer requires 2 bitcoin transactions, load on the bitcoin network and block size will be at least double that of the many proposals requiring just one transaction.

"Tokeda favors the storage of plain, i.e. non-hashed, data. Hashing metadata diminishes the burden for the Bitcoin blockchain, however, it immediately creates another problem: the only way to reliably recover the hashed metadata is to have a layer 2 of some kind involved." (page 6)

In other words, all Tokeda data is stored on-chain. Therefore network and blockchain storage grow at least linearly with Tokeda usage, similar to Group tokenization. The only reduced storage is in pruning token data from the UTXO set *of nodes that don't care about some tokens.* Tokeda seems to assume that tokens will not be popular and from that claims a storage benefit for the UTXO set only, at a not-quantified (because the document does not fully specify the relevant transactions) but likely double or more additional overhead for blockchain data.

Now let us quantify the UTXO set storage benefit claim, made on page 14:
"The Tokeda scheme relies on a one-time miner fee (or rather two-time considering the transaction relay) to make the miner profitably process the token metadata. However, this process is one time cost as well - all Tokeda data being fully prunable - unless the miner decides to persists some UTX entries."

But on page 16, the basic Tokeda ownership scheme is finally described as follows:

"A token holder is an agent who has control over an UTXO entry that is decorated with a Tokeda metadata. The metadata is written in the UTX set. This input script identifies the issuer, while the metadata details the state of the token."

Since Tokeda ownership is controlled by UTXO entries, I guess we need a UTXO per Tokeda entry. The only prunable data seems to be the Tokeda metadata, which is equivalent to the Group identifier in the Group tokenization proposal. Althogh the statement of "all Tokeda data being fully prunable" may be pedantically true but we all understand that these UTXO would not exist if they were not carrying tokens. Which is larger, the required UTXO or the prunable data?

Note that the smallest reasonable UTXO entry contains at least 35 bytes (a pubkey or script hash (160 bits) and some code (3 8-bit instructions minimum), an amount (64), and block height (32)), and it needs to be looked up via transaction identifier (32 bytes) and index (4 bytes). This therefore requires, at a minimum, 71 bytes.

Group tokenization proposes 32 byte token identifiers which would be included in each UTXO. In that case, Tokeda's pruned UTXO set is therefore 31% smaller than Group tokenization. But it would make sense for Group tokenization nodes to internally store token identifiers using a shorter identifier. For example, nodes could store the first 4 billion groups created in a 32 bit field. In this case, Tokeda's pruned UTXO set is only 5.3% smaller than Group tokenization.

The existence of the UTXO is much more important than the pruning of some token metadata.

Let's run some numbers for a don't-care-about-tokens, full history full node:

The UTXO set is currently 1.54% of the blockchain data (the blockchain data is currently at 143GB and the UTXO set at 2.2GB). Let us assume that approximate ratio of history to UTXO for tokens, and propose a similar quantity of history creates another 1GB of tokenized, prunable UTXO. In the Group tokenization case, this results in an additional 65GB of history and a 5.3% bigger UTXO, or 1.053GB, for a total of 143+2.2+65+1.053 or 211.25GB of storage (or 211.65GB for the uncompressed token identifier case). In the Tokeda case, 2 transactions are required to make a single transfer. This means that 1GB of tokenized, prunable UTXO results in 130GB of history and 1GB UTXO, for a total of 143+2.2+130+1 or 276.2GB of storage.

Tokeda's need to make 2 blockchain transactions to effect a single transfer make it significantly less storage-efficient than Group tokenization. Even if the poorly defined "direct metadata publishing" technique is used (discussed below), possibly allowing 1 blockchain transaction per tokeda transfer, Tokeda's UTXO data set would be about 5% smaller than Group tokenization, which is negligible compared to blockchain storage requirements.

Since the transaction data is not prunable as they traverse the network, Tokeda's network utilization does not benefit from pruning, and it still suffers from its 2 transactions-per-transfer problem. The Tokeda network utilization numbers are even worse than its storage numbers.

Specification Fantasy

The Tokeda proposal is organized with 16 pages that makes general statements about Tokeda, describing token philosophy and use cases and how Tokeda fulfills them. It then finishes with 3 pages of specification (pages 17 to 20), and finally includes an appendix with 10 pages of extended use cases.

This document is therefore organized such that effect (Tokeda's benefits in certain use cases) precedes cause (Tokeda's actual architecture). The sequential reader therefore has no ability to verify the author's promises of Tokeda functionality as s/he encounters them and instead must take the assertions made on faith, or attempt to skip forward and find the relevant specification.

This document structure is not "best practice", to put it kindly, because it inevitably results in promises that are in conflict with or not explained by the specification, ultimately resulting in a product that cannot meet its promises (vaporware).

Since the document has been marked "early draft", you may consider it unfair to point out unfinished portions. However, by leaving Tokeda only partially specified, it can be used to shoot down other proposals via unfulfilled promises, and its limitations are hard to pin down because every criticism can be met with an unproven claim. This is a common argumentation technique and the key technique is to delay concrete specification. This Tokeda document has been unchanged since April 30.

"Specification Fantasy" Examples

Token Capabilities (page 16)

The specification states that "A token holder is an agent who has control over an UTXO entry that is decorated with a Tokeda metadata". So if the issuer doesn't control that UTXO, how can it:

3. Issuer can revoke existing units [I'm presuming this means units it doesn't own based on the word "revoke"]
4. Issuer can callback tokens
7. Issuer can update tokens

Actually, the mechanism to implement every single one of the capabilities specified except 9 (transfer) is not defined in the specification, so they should all be considered fantasy. However, given the broadly specified architecture we can reasonably assume some capabilities are possible (such as minting). But the capabilities indicated above seem to be impossible based on the proposed UTXO architecture where the token holder controls the UTXO but somehow the issuer can magically spend it to revoke, callback or update the token.

Additionally note that capability 8 and 9 are also not possible:

8. User can revoke existing units
9. User can transfer existing units

The user does not have these capabilities. As per the Tokeda spec, cooperation between the user and issuer is needed to execute these capabilities. The user must first send the UTXO to the issuer who then can revoke or transfer the units (or keep them, send them back to the user, etc).

“Direct metadata publishing”

Tokeda proposes a technique on page 15 where a user submits a transaction directly to the issuer for signing, receives a response and publishes it to the blockchain. No actual specification of this protocol exists. The document suggests two techniques where the issuer can be defrauded by the user, and suggests a deposit that the issuer can presumably take in case of fraud. This limits the value transferred via the direct mode to less than the deposit, ignores fluctuating values, and ignores the situation where the issuer fraudulently steals all deposits.

Additionally, using this mode, the issuer can deny transfers in a manner that is impossible for third parties to detect, which is in conflict with the “trust-but-verify” security promise. This is likely why the mode is not mentioned in any other part of the document, including the specification section.

A possible solution would be a scheme where the user first attempts “direct metadata publishing” and subsequently attempts 2 phase publishing. However, a quality specification does not make the reader fill in its gaps.

Due to the lack of detail, attacks, and conflict with the foundational “trust-but-verify” premise, this mode is given secondary consideration in this document.

“Tokeda also offers market signaling mechanisms such as proof-of-worth to let the market differentiate the valuable tokens from the lesser ones” (p 4)

Additional paragraphs are located on page 13 that clarify proof-of-worth being the total mining fees consumed by the token. The well known problem of miners paying fees to themselves is mentioned but not solved. The known solution (e.g. fee pool) involves a hard fork, so Tokeda will either require a hard fork, or this document is engaging in specification fantasy.

Additionally, the document makes the assumption that you can derive a measure of token worth by summing transaction fees. In other words, that transaction fees are somehow related to token value, yet by actually looking at the fee calculation code (and examining the “miner problem” theoretically), one can show that transaction fees are actually (and should be) proportional to the size in bytes of the transaction. In practice we have seen that transaction fees are related to transaction size and block space competition. In short, the proposed approach to calculating proof-of-worth is completely flawed.

Exchange

Atomic exchange is not defined. It seems likely that atomic exchange (Alice sends Bob token X and Bob sends Alice token Y atomically) would be very difficult since token transfer is itself not atomic, unless Trent is the issuer for both tokens.

Even if Trent is the issuer, Alice and Bob would have to communicate their desire for an atomic exchange through some undefined protocol.

Is it possible to trust-trent-but-otherwise-atomically exchange a Tokeda token for BCH? Perhaps. It is all left to reader speculation. Alice would send her tokens to Trent, indicating Bob as her destination, and Bob would send Trent BCH, indicating Alice as his BCH destination. Trent needs specific software to handle this operation.

If Trent was capable of sending tokens to a non-trivial output script, could a technique similar to cross-chain atomic swaps (but on the same chain) be used to exchange tokens? Perhaps, but these techniques require special scripts, so again Trent needs to be specifically coded for the feature, including a signaling protocol. Also these scripts typically require transactions to be spent before a timeout or the other party can recover funds. So Alice would need to spend Trent’s output to herself, through Trent, using another 2 transactions.

Issuer Best Practices -- Key Rotation

Issuer key compromise is an ever present risk. Even if multisig addresses are used, best practices is to allow rotation of key material in the case where a Tokeda signatory changes, or due to personnel changes within a signatory agency.

Tokeda issuer identity seems to be based on some kind of key (exactly what it is is not defined), but *“The issuer is identified through inspection of the input script.” (page 6)* The only unspoofable key in the input script is a pubkey and associated signature, so my assessment is that the Tokeda issuer identity is probably intended to be a bitcoin pubkey.

“The stateless wallet compatibility of Tokeda is ensured because a user only needs to persist two elements to transact with the token: its own secret seed and the issuer identity.” (page 5)

Changing this identity means changing it in all end-user wallets. This may be difficult. While claiming key rotation as a use case, the tokeda document acknowledges the wallet problem in a footnote on page 14 without actually presenting a solution.

Issuer Best Practices -- “Cold” key storage

Tokeda issuer keys must be used to sign every transfer so offline key storage seems to be impossible. However, if this same key is compromised, it seems that it can be used for any Tokeda operation including minting new tokens.

Safer architectures allow key role separation -- high value keys are not needed for every day operations.

Questions

(the questions seem endless but here are a few)

What if...

1. Alice needs change (in tokens)?
2. Alice sends a token-decorated UTXO to someone who is not Trent?
3. Alice wants to combine UTXOs?
4. Alice’s request to send to Bob is denied by Trent?
5. Trent makes a mistake?
6. Validators spot a mistake? Is there a protocol to communicate that?
7. Validators lie about a mistake? (Are fraud proofs possible)

How is...

8. Bob identified in the Tokeda metadata?
9. Trent choosing the output script when sending to Bob?
10. The “inner payload” defined for anything but the initial token creation?

11. Trent going to communicate transfer problems to Alice (and Bob)?
12. Alice going to get the BCH in a Tokeda UTXO back?

Conclusion

Although much of Tokeda is completely unspecified, it seems to propose a system where token holders control a UTXO that should only be spent to the issuer, who has the opportunity to apply arbitrary policy before forwarding the spend to its actual destination. It is therefore an authority-based, SPV capable system. However it ineptly deploys the power of authority-based systems resulting in problems easily solved by other authority systems.

By placing its UTXO on the blockchain, and requiring 2 transactions per transfer, it compares very unfavorably with respect to scalability with many other token proposals, including the permissionless Group tokenization. Authority-based tokens should be able to do much better in their ability to shard the UTXO. For example authority-based extension block systems such as FSHblocks (talk, spec) can move all token transfers completely off-chain -- out of history AND UTXO. The only on-chain transfers required are those that are actually moving BCH value between the BCH blockchain and the extension block.

Tokeda's two transaction "referred transfer" model likely severely limits token interaction with BCH, scripts, and other tokens. Every protocol (for example, onchain betting or probabilistic payments) will likely need to be explicitly programmed into and overseen by the issuing agent. Additionally, by storing user balances in a user-controlled UTXO, it cannot implement many features common to authority based approaches, like recalling tokens.

Given its lack of interactivity with BCH and authority-based architecture, there seems to be no reason whatsoever for Tokeda's implementation on a blockchain.