Cross-posted from a slightly more live version on github.

Off-chain Issuable Tokens

Some tokens have a central issuing authority, like for coins in an online game, which may only gain interesting value meriting an on-chain transaction once they accrue in a significant quantity.

In cases like this, issuance can be performed in an offchain manner that maintains on-chain trustlessness.

State channels require an initial deposit because it is assumed that the sender has a limited quantity of tokens. In the case of a centralized issuer, you have an implicit infinite deposit of tokens, and this allows tokens to be counterfactually issued with a simple signed message.

An example issuance message, signed by the issuer might look like this:

struct IssuanceMessage { address recipient; uint amount; uint issuanceBlock; // could also use time stamp }

The issuanceBlock or time stamp is used so that once a message is used to mint and redeem some tokens, no message before that redemption block height is valid. This ensures redeemers are always incentivized to only try to redeem their most recent message, and cannot replay messages.

This requires the issuer to recognize when tokens are redeemed, and restart their amount

counter for messages sent to that user.

A sample extension to the token interface would be a method such as:

function redeemPromisedTokens(bytes payload, uint8 v, bytes32 r, bytes32 s) public returns (bool);

Pseudolidity Implementation

contract OffchainIssuableToken is MintableToken {

mapping (address => uint) latestWithdrawBlock;

struct IssuanceMessage { address recipient; uint amount; uint issuanceBlock; }

function redeemPromisedTokens(bytes payload, uint8 v, bytes32 r, bytes32 s) public returns (bool) { assert(verify(payload, v, r, s));

IssuanceMessage memory msg = parse(payload, v, r, s);

// Ensure user submits the latest message they have
// to prevent replay attacks:
assert(msg.issuanceBlock > latestWithdrawBlock[msg.sender]);
latestWithdrawBlock[msg.sender] = this.blockNumber;

mint(msg.recipient, msg.amount);

} }

Wallet Implementation

Wallets could store these messages and display them like tokens, and they would not need to be privately held unless privacy were a goal, because the only thing the message could be used for would be issuing these tokens.

Use Cases

An AI company wants to reward online participants for training its web interface, and wants to issue tokens for each attempt without an on-chain transaction.

Personal currencies that are backed primarily by an individual's trust, so the threat of inflation is already accounted for in the risk of holding that coin.

An organization wants to crowdfund in person, in cash, and give out certificates for token redemption without bothering with on-chain transactions, especially when (like with some crowd-funding ventures) the project may not complete, and the tokens may not be worth redeeming anyway.

Casual or localized token experiments: Selling a gift card

This pattern also works when offline, so even in a low-internet environment, if a person or entity has the ability to sign minting messages for a token that has a known value, they could sign these messages as a way to pay someone “once they’re online”.

Possible other features

The signed message could include a reference to the issuing contract, to allow the issuing key to issue multiple different tokens.

There could be a redeemAndSend

method so that a user can both redeem their tokens and send some of them in a single transaction.

The token contract is given the power to verify signatures and enforce issuance however it wants. There could be a list of approved issuers, and individual issuers could even have issuance limits to tranche liability of compromised issuing keys.

If a withdraw period + unainamous signatures were added (like in a traditional state channel), then the tokens issued could also be bi-directionally exchanged, without ever making an on-chain transaction.