OST: An Overlay Protocol for Scheduling Multiprocess Decentralised Computation

OpenST Foundation Ltd.

December 29, 2017, v0.9

1 Introduction

The advent of open, decentralised blockchain systems has demonstrated that consensus engines driven by market forces can support an ongoing, crypto-economically secured transaction ledger. Most notably Bitcoin and Ethereum have successfully implemented a respective stateless asset and stateful account model on top of Proof-of-Work consensus engines. Both systems charge a variable transaction fee to include and process transactions into the ledger and to dynamically modulate network demand given constraint transaction throughput capacity. More strongly so, it has been shown [VerifiersDilemma ref] that for Proof-of-Work the computational *efficiency* must be low, i.e. the amount of useful computation done in a set of transactions must be small compared to the amount of hashing power expended to secure that set into a block. Both Bitcoin and Ethereum implement a limit for the useful computational expenditure a valid block can contain measured in bytes or gas respectively. As Proof-of-Work maximises its security by maximising the amount of computational effort spent, both systems increase the transaction throughput capacity over time. The cost for the same

[TBC]

2 Cores

Let a *core* d be a deterministic, finite state machine with a merklelized state root after each state transition upon which one can implement the OST protocol presented. For efficiency reasons we assume that the core groups state transitions into blocks and we will consider in this paper all cores to be a blockchain network. A blockchain network additionally has a consensus engine with rules r_d to reach consensus on a linear sequence of blocks.

Consider two cores d_1 and d_2 that each produce blocks according to their consensus rules. The cores can produce blocks independently. Assume a node that fully verifies both cores. The node can report the block header of d_1 to d_2 and reversely d_2 to d_1.

3 Transposing Tokens

4 Account Sessions

An account allows a user to manage her devices. Devices in turn can be used to create and revoke sessions, and redeem branded tokens. A session combines a session key and a set of single-use, cryptographically linked secrets which the session key can include to authorize new transactions on behalf of the account with.

A session has spending limits, and a single secret has smaller spending limits. Spending limits can be a combination of amount restrictions, expiration or frequency constraints and are enforced by the account smart contract. The session spending limits can be smaller than the product of the number of secrets and the secret spending limits.

To create a session an application client privately generates a random 256-bit seed and produces n iteratively keccak256 hash images of this seed. The last generated image we label s_0, the pre-image to s_0, s_1 and so on:

$$s_i = \text{keccak256}(s_{i+1}) \qquad (1)$$
$$s_n \text{ is the initial random seed}$$

where $i \in [0, n-1]$. The application server generates a new session key and the server is responsible for funding the session key with Simple Token Prime as the base token to pay for gas costs. The application server and the application client exchange the address for the session key and s_0. The application client generates a crypto-image with the address and the initialisation secret s_0 and presents it to the user. The application server pushes the session address and the initialisation secret to the device for proposal. Within Simple Token Wallet (or other wallet integrations) that manages the device private key the user can visually verify the same crypto-image, set spending limits and approve the session by signing with the device key held in the wallet activating the session.

Within an active session the session key can sign transactions to spend from the account if it presents the next pre-image secret for the session and within the spending limits of the secret and the session.