I’ve coded a Solidity implementation of @rmsams Seignorage Shares. The contract is up and running on the Rinkeby testnet.

SeignorageController:

https://rinkeby.etherscan.io/address/0xcb8c6d00c7d303b6d24794cf960659c4682c3e47

Shares:

https://rinkeby.etherscan.io/address/0x75071ac8edfd1a060b3d7298cc89f5fac44a1994

Coins:

https://rinkeby.etherscan.io/address/0x1bdd0c76810e17475a4d0303aa1777f5143d18ff

The source code is uploaded to Etherscan under the inidividual contracts and also available on Github:

GitHub

k26dr/seignorage-shares

Solidity implementation of Robert Sams Seignorage Shares - k26dr/seignorage-shares

Some implementation notes:

- A single oracle (me) has control of the price feed for now. The next step is to change this to either a multi-oracle or Schelling point scheme.
- The MINT_CONSTANT

determines the the percentage change in supply for a percentage change in price. Currently a 1% change in price leads to 0.1% change in supply per cycle.

- The CYCLE_INTERVAL

is currently set to 2000 blocks (~10 hours). An auction is held every cycle for newly printed shares/coins.

- The auction design is pulled from the EOS crowdsale. Users send their orders to a common pool. The sum total of all orders for a cycle is divided by the printed amount of coins/shares to determine the exchange price. Arbitrageurs are incentivized to place orders and bid up the price until the auction exchange price reaches the market price.
- The auction method requires the number of shares to mint to be known ahead of time, so two oracles are required (share price + coin price) instead of one like in the original white paper.

If you’re interested in the project and want some coins/shares on the testnet, shoot me your address in a reply.