JohnLawCoin:
A Stablecoin with an Algorithmic Central Bank

Kentaro Hara*

2020 Jan 23

Abstract

JohnLawCoin is a stablecoin realized by an **Algorithmic Central Bank (ACB)**. The system is fully decentralized and there is truly no gatekeeper. No gatekeeper means there is no entity to be regulated.

JohnLawCoin is a real-world experiment to verify the following assumption: It is possible to stabilize the currency price with fully algorithmically defined monetary policies without holding any collateral.

If JohnLawCoin is successful and proves the assumption is correct, it will provide interesting insights for both non-fiat currencies and fiat currencies; i.e., 1) there is a way for non-fiat cryptocurrencies to stabilize their currency price without having any gatekeeper, and 2) there is a way for central banks of developing countries to implement a fixed exchange rate system without holding adequate USD reserves. This will upgrade human's understanding about money.

1 Motivation

1.1 Stablecoin

The high volatility of Bitcoin, Ethereum and other cryptocurrencies has limited their use cases in real-world transactions. For example, the price of ETH (Ethereum's currency) dropped from $1300 to $84 in 2018. Cryptocurrencies are recognized as speculative assets, not money. In theory, the main functions of money are distinguished as: a medium of exchange, a unit of account, and a store of value [1, 2]. For cryptocurrencies to realize these functions and become money, price stability is a mandatory requirement. Stablecoins have explored ways to ensure the price stability, including Tether [3], Libra / Diem [4], MakerDAO [5] and Synthetix [6]). JohnLawCoin is one of the efforts.

1.2 No entity to be regulated

The fundamental difference between JohnLawCoin and other stablecoins is that JohnLawCoin has no gatekeeper. A gatekeeper means an entity that has control and intention about the ecosystem. For example, Tether is issued by Tether Limited, which is subject to be regulated as a gatekeeper when something inappropriate happens in the ecosystem. On the other hand, JohnLawCoin is fully decentralized by defining open protocols that stabilize the currency price algorithmically. This eliminates the need to create a gatekeeper in the ecosystem. There is no entity to be regulated.

*Twitter: @johnlawcoin, GitHub: https://github.com/xharaken/john-law-coin. Opinions are my own and not the views of my employer.

1.3 No collateral

JohnLawCoin stabilizes the currency price without holding any collateral (e.g., USD, gold, other cryptocurrencies). This is important to eliminate a gatekeeper that holds the collateral from the ecosystem.

In ancient times [1, 2], it was believed that the value of money derives from the purchasing power of the commodity upon which it is based. This is known as metalism. Later the metalism was replaced with chartalism, where it was believed that the value of money derives from the sovereign power to enforce the circulation and levy taxes. The shift from metalism to chartalism was a fundamental improvement because chartalism enabled the government to issue more money than the amount of metal the government held. Even in this case, however, it was long believed that the government has to hold gold reserves to back the circulating money. This is the gold standard, which was a standard currency system in the 19th and 20th centuries. It was 1973 when the United States unilaterally terminated the convertibility of USD to gold by bringing the Bretton Woods system to an end. At this point, which was only 50 years ago, humans finally understood that floating fiat currencies that are not backed by gold work. Humans are slowly getting to understand that money works as money without being backed by collateral.

As Katsuhito Iwai pointed out [7, 8], money works as money simply because it is used as money. Money is accepted as money by everybody merely because it is accepted as money by everybody else. This infinite incentive recursion is the origin of the value of money. Collateral like gold and the sovereign power is indeed useful to establish the infinite incentive recursion but is not the essence to make something work as money.

However, it seems like that existing stablecoins still believe the myth that money must be backed by collateral (e.g., Tether, Libra, Maker) and are hitting challenges:

- Tether [3] is backed by USD. Tether Limited works as a gatekeeper and holds the USD reserves. However, it is pointed out that Tether is controversial because of the company's failure to provide a promised audit about the USD reserves and also its alleged role in manipulating the price of Bitcoin. Tether's centralized model brings counterparty risks.
- Libra [4] is backed by various fiat currencies and financial assets. The Libra Association works as a gatekeeper and holds the assets. Even though the Libra Association is established as an open foundation, it was regulated. The fact that there is a gatekeeper to hold collateral enabled the governments to regulate the gatekeeper.
- Maker [5] uses ETH as reserves. However, the price of ETH is unstable and Maker needs to ask users to keep a 150+% collateralization ratio for ETH (e.g., when a user reserves 150 ETHs, the user can issue up to 100 coins). If the collateralization ratio drops to less than 150%, the reserved ETHs are forced to default and liquidated. This actually happened when the price of ETH dropped from $1300 to $84 in 2018 and many users lost their reserved ETHs.

These complexities come from the collateral. The goal of JohnLawCoin is to demonstrate that "money must be backed by collateral" is a myth. JohnLawCoin is a real-world experiment to verify the assumption that **there is a way to establish the infinite incentive recursion and stabilize the currency price without holding any collateral**.

1.4 Modern Monetary Theory

JohnLawCoin is inspired by Modern Monetary Theory (MMT) [9, 10]. Specifically, JohnLawCoin is designed based on the following monetary principles of MMT:[1]

[1] MMT is controversial when it comes to its political implications about fiscal policies and taxing [11], but digging into the political implications is out of the scope of this experiment. In the first place, JohnLawCoin does not have any gatekeeper that can enforce fiscal policies or taxes. The author does not want to highlight JohnLawCoin's dependencies on MMT too much because MMT is recognized as "heterodox" economics due to the controversial political implications. Even if MMT fails, it does not mean JohnLawCoin fails. JohnLawCoin only depends on the monetary principles [MMT-I], [MMT-II] and [MMT-III], which do not contradict the standard economic theories.

[**MMT-I**] A central bank that issues its own fiat currency is never forced to default on debt denominated by the fiat currency because the central bank can always redeem the debt by creating money.

[**MMT-II**] The money creation and the increased balance of the debt are not a problem as long as they do not lead to inflation.

[**MMT-III**] The central bank does not need to issue debt (and borrow money from private sectors) to create money because the central bank can create money from scratch. Debt is not a tool for financing the money creation but is a tool to control the currency price.

2 Value proposition

JohnLawCoin is a real-world experiment to upgrade human's understanding about money and collateral. If JohnLawCoin is successful about stabilizing the currency price, it will provide new insights to both non-fiat currencies and fiat currencies:

- **Non-fiat cryptocurrencies can use the algorithm to stabilize their currency price without holding collateral**. This releases the non-fiat cryptocurrencies from the counterparty risks and the regulation risks.
- **Real-world central banks of developing countries can use the algorithm to implement a fixed exchange rate system without holding adequate USD reserves**. This could have helped to prevent historical financial crises [2] caused by speculative attacks that attempted to exhaust government's USD reserves and thus break the fixed exchange rate (e.g., Black Wednesday in 1992, Asian financial crisis in 1997). JohnLawCoin will have interesting implications to the international currency systems.
- **Real-world central banks can use the algorithm to implement rule-based monetary policies**.

3 Related work

Historically, there were a couple of efforts that attempted to implement stablecoins with no collateral. Examples are BasisCoin [12], Seigniorage Shares [13] and Kowala [14]. BasisCoin is the most popular and invested one. BasisCoin secured $133 million in funding but shut down before launching because the company could not find any way to pass regulatory constraints.

JohnLawCoin has commonalities with BasisCoin in the sense that it uses bonds to control the total coin supply. However, there is a fundamental difference in the launch strategy. In case of BasisCoin, a start-up company attempted to launch the system and was in charge of the issued tokens and bonds. Thus the company was regulated as a gatekeeper. In case of JohnLawCoin, it is fully decentralized and there is no entity to be regulated. Another difference is in the protocol. BasisCoin used three types of tokens: shares, coins and bonds. JohnLawCoin uses only two types of tokens: coins and bonds. JohnLawCoin's protocol is simpler and easier to understand.

To the best of the author's knowledge, there is no large-scale real-world experiment that has explored stablecoins with no collateral. BasisCoin looked like the most promising experiment but shut down before launching.

4 Algorithm

4.1 Overview

First of all, JohnLawCoin needs to define what its value is bound to. In theory, this can be anything (e.g., USD, a currency basket, Consumer Price Index), but JohnLawCoin binds one coin to one USD.

JohnLawCoin consists of the four components:

Coins: The coins are the core tokens intended to be used as a medium of exchange. The goal of JohnLawCoin is to stabilize the currency price of the coins (i.e., 1 coin = 1 USD).

Bonds: The bonds are used to expand and contract the total coin supply and thus adjust the currency price of the coins. The bonds are designed as zero-coupon bonds where the annual interest rate is set to R. One bond is redeemed for B_{redemp} coins on the redemption date. Let T be the redemption period (measured in days) and B_{issue} be the bond issue price. R, T, B_{redemp} and B_{issue} meet $B_{\text{issue}}(1+R)^{365/T} = B_{\text{redemp}}$.

Oracle: The oracle is a mechanism to determine the JohnLawCoin / USD exchange rate in a fully decentralized manner.

Algorithmic Central Bank (ACB): The ACB obtains the JohnLawCoin / USD exchange rate from the oracle. The ACB expands and contracts the total coin supply so that the exchange rate becomes 1.0.

The following sections describe how the oracle and the ACB work.

4.2 Oracle

Since the information about the JohnLawCoin / USD exchange rate is external to a blockchain on which the ACB runs, the system needs a fully decentralized mechanism to feed the external information to the blockchain. This mechanism is called an oracle and multiple solutions have been proposed [15, 16]. JohnLawCoin uses a combination of the commit-reveal scheme [17] and Schelling Point [18]. Table 1 shows an overview of how it works.

Table 1: An overview of how the oracle works.

1. Commit phase	A voter deposits $D\%$ of their coin balance. The voter commits hash(*quantized exchange rate*, *salt*).
2. Reveal phase	The voter reveals *quantized exchange rate* and *salt*.
3. Reclaim phase	Voters who voted for the "truth" exchange rate or its surroundings can reclaim the coins they deposited in the commit phase. In addition, voters who voted for the "truth" exchange rate can get a reward.

4.2.1 Commit phase

In the commit phase, anyone can vote for what they think the JohnLawCoin / USD exchange rate is. The voter is expected to obtain the information from real-world currency exchangers. Since the oracle accepts only discrete values (e.g., 0.7, 0.8, ..., 1.2, 1.3), the voter needs to quantize the exchange rate to the closest discrete value accepted by the oracle. The voter submits hash(*quantized exchange rate*, *salt*) to the oracle. The hash function is public and shared with all the voters. The *quantized exchange rate* and the *salt* are secret to each voter. The voter deposits $D\%$ of their coin balance to the oracle.

4.2.2 Reveal phase

In the reveal phase, the voter reveals the *quantized exchange rate* and the *salt* they used in the commit phase. The oracle weights the revealed votes by the amount of the coins deposited in the commit phase and calculates the mode of the votes. The mode is defined as the "truth" exchange rate. Due to the weighting, the more coins you possess, the more power your vote has.

4.2.3 Reclaim phase

In the reclaim phase, the voters who voted for the "truth" exchange rate or its surroundings can reclaim the coins they deposited in the commit phase. For example, if the "truth" exchange rate is 1.1, the voters who voted for 1.0, 1.1 and 1.2 can reclaim the coins they deposited. Other voters lose the coins they deposited.

In addition, the "winner" voters who voted for the "truth" exchange rate can get a reward. The source of the reward is the coins lost by the other voters and the coins the ACB decided to mint to expand the total coin supply (explained below). C% of the reward is evenly distributed to the "winner" voters. The rest of the reward is distributed to the "winner" voters in proportion to the coins they deposited.[2][3]

This reward mechanism incentivizes voters to vote for the "truth" exchange rate.[4][5] The voting to the oracle can be viewed as coin mining but remember that the oracle is neutral to the total coin supply. The oracle does not mint any coins. The oracle only redistributes the deposited coins and the coins the ACB decided to mint to expand the total coin supply.

4.3 ACB

4.3.1 Algorithm

The ACB obtains the quantized exchange rate from the oracle. The ACB adjusts the total coin supply so that the exchange rate becomes 1.0 with the following algorithm.

---

struct ACB:
```
    # The total coin supply
    int coin_supply;
    # The mapping from a user to their coin balance
    mapping<address, int> coins;
    # The total bond supply
    int bond_supply;
    # The mapping from a pair of user and the bond redemption timestamp
    # to the bond balance
    mapping<(address, int), int> bonds;
    # If bond_budget is positive, the ACB is ready to issue bond_budget bonds.
    # If bond_budget is negative, the ACB is ready to redeem bond_budget bonds.
```

[2] You may think C can be 0 (i.e., all of the reward is distributed to the "winner" voters in proportion to the coins they deposited). However, it is important to set a positive value to C to bootstrap the system. Initially, all voters except the genesis account have 0 coins. If C is 0, the voters do not have any incentive to contribute to the oracle because they can get no reward by doing so (even worse, they lose the transaction fee for Ethereum). C needs to be a positive value to bootstrap the system.

[3] C should be set to a small value. If C is large, it will work as an incentive to create many dummy accounts. This may happen with a small C to some extent but it will not have a terrible impact to the system. The ACB's ability of controlling the total coin supply and adjusting the currency price is not affected by how many accounts are created in the system.

[4] In essence, the oracle creates a Keynesian beauty contest [19].

[5] Coin holders are incentivized to increase the value of the coins. What happens if a subset of users who hold 50+% coins intentionally vote for a wrong exchange rate to let the ACB mint or burn coins as they like? Two things should be noted. First, no decentralized algorithm works if 50+% of the participants behave dishonestly (so JohnLawCoin assumes 50+% of the participants behave honestly). Second, if they do that, JohnLawCoin will lose trust and the value of the coins will drop. The coin holders are not likely to have incentives to manipulate the exchange rate dishonestly.

```
    int bond_budget;
    # The oracle
    Oracle oracle;

# The ACB calls this function every period.
function AdjustCoinSupply(ACB acb):
    float exchange_rate = acb.oracle.GetExchangeRate();
    # Calculate the amount of coins to be minted or burned
    int delta = int(k * acb.coin_supply * (exchange_rate - 1.0));
    if delta == 0:
        acb.bond_budget = 0;
    else if delta > 0:
        int count = delta / B_redemp;
        if count <= acb.bond_supply:
            # If there are enough bonds to redeem, increase the total coin
            # supply by redeeming bonds
            acb.bond_budget = -count;
        else:
            # Otherwise, redeem all the issued bonds
            acb.bond_budget = -acb.bond_supply;
            int mint = (count - acb.bond_supply) * B_redemp;
            acb.coin_supply = acb.coin_supply + mint;
            # Provide the remaining coins to the oracle as a reward
            acb.oracle.provideAsReward(mint);
    else:
        # Issue new bonds to decrease the total coin supply.
        acb.bond_budget = -delta / B_issue;

# The ACB calls this function when user requested to purchase count bonds.
function IssueBonds(ACB acb, address user, int count):
    if acb.bond_budget < count or count <= 0:
        return;
    int amount = count * B_issue;
    if acb.coins[user] < amount:
        return;

    # The redemption timestamp of the bonds.
    int redemption = CurrentTimestamp() + T;

    # Issue new bonds
    acb.bond_supply = acb.bond_supply + count;
    acb.bonds[(user, redemption)] = acb.bonds[(user, redemption)] + count;
    acb.bond_budget = acb.bond_budget - count;

    # Burn the corresponding coins.
    acb.coin_supply = acb.coin_supply - amount;
    acb.coins[user] = acb.coins[user] - amount;

# The ACB calls this function when user requested to redeem bonds whose redemption
# timestamp is redemption.
function RedeemBonds(ACB acb, address user, int redemption):
    int count = acb.bonds[(user, redemption)];
    if redemption > CurrentTimestamp():
        # If the bonds have not yet hit their redemption timestamp, the ACB
```

```
    # accepts the redemption as long as the budget is negative.
    if acb.bond_budget >= 0:
        continue;
    if count > −acb.bond_budget:
        count = −acb.bond_budget;
# Burn the redeemed bonds.
acb.bond_supply = acb.bond_supply − count;
acb.bonds[(user, redemption)] = acb.bonds[(user, redemption)] − count;
acb.bond_budget = acb.bond_budget + count;

# Mint the corresponding coins.
int amount = count ∗ B_redemp;
acb.coin_supply = acb.coin_supply + amount;
acb.coins[user] = acb.coins[user] + amount;
```

4.3.2 Total coin supply

Let M be the total coin supply and E be the quantized exchange rate obtained from the oracle; i.e., one coin is convertible to E USDs.

If $E > 1$, the ACB increases the total coin supply by $k\,M\,(E-1)$. If $E < 1$, the ACB decreases the total coin supply by $k\,M\,(1-E)$. k $(0 < k < 1)$ is a damping factor to avoid making an overly aggressive change to the total coin supply. According to the Quantity Theory of Money [20], k should be set to 1. For example, if E is 2.0, the Quantity Theory of Money states that the total coin supply should be doubled and then E is adjusted to 1.0. However, the Quantity Theory of Money oversimplifies the reality where velocity of money is not stable. Therefore the ACB sets k to a lower value to avoid minting or burning too many coins in one period.

4.3.3 Minting / burning coins

Now the ACB knows how many coins should be minted (when $E > 1$) or burned (when $E < 1$). The next question is how the ACB mints or burns the coins.

The ACB mints coins by redeeming issued bonds regardless of their redemption dates. Specifically, the ACB redeems $k\,M\,(E-1)/B_{\text{redemp}}$ bonds regardless of their redemption dates.[6] If $k\,M\,(E-1)/B_{\text{redemp}}$ exceeds the number of issued bonds, the ACB mints the remaining coins and provides them to the oracle as a reward. As described above, the oracle distributes the reward to voters who voted for the "truth" exchange rate.

The ACB burns coins by issuing new bonds. Specifically, the ACB issues $k\,M\,(1-E)/B_{\text{issue}}$ bonds.

In this way, the ACB increases / decreases the total coin supply by redeeming / issuing bonds. The redeem / issue operations do not require an open market. The ACB redeems a bond at a price of B_{redemp} and issues a bond at a price of B_{issue}. The ACB can always redeem / issue bonds without relying on an open market at all.[7]

5 Incentive structure

For the described algorithm to work, the following condition must be met:

[**Condition**] The ACB can find users who are willing to buy issued bonds.

[6] The ACB does not need to worry about what bonds should be redeemed first (e.g., first-issued-first-redeemed) because it is bondholder's responsibility to request the redemption. The ACB only needs to redeem bonds as requested until $k\,M\,(E-1)/B_{\text{redemp}}$ bonds are redeemed. Note that bondholders are incentivized to redeem bonds as soon as possible to get the interest rate.

[7] An open market may emerge but that's a separate discussion.

If the [Condition] is met, the algorithm works. If the [Condition] is not met, the algorithm does not work because the ACB cannot decrease the total coin supply by issuing bonds. This section analyzes when the [Condition] is met or not met using Domar's theorem [21].

Let R be the annual interest rate of the bond and G be the annual growth of the economy. The total coin supply is expected to grow with the annual growth rate of G.

It is important to understand that no matter what happens with G and R, the following statements hold:

(a) The ACB can redeem one bond for B_{redemp} coins when the redemption date comes.

(b) The bondholder can get an interest with the annual interest rate of R.

(c) If the [Condition] is met, inflation does not happen. One coin is convertible to one USD when the redemption date comes.

(a) is obvious. According to [MMT-I], a central bank that issues its own fiat currency is never forced to default on debt denominated by the fiat currency because the central bank can always redeem the debt by minting the fiat currency. (b) is obvious per the bond definition. (c) holds because if the [Condition] is met, the ACB can continue running the algorithm and the algorithm adjusts the JohnLawCoin / USD exchange rate toward 1.0 (i.e., inflation does not happen).

Therefore, rational users are incentivized to buy bonds if the users believe the [Condition] is met when the redemption date comes. The question is when the users believe the [Condition] is met. The following three cases need to be analyzed: $G > R$, $G = R$, and $G < R$.

Case 1: $G > R$. In this case, the balance of the issued bonds decreases (and may go down to zero) over time because the ACB needs to redeem bonds faster than issuing bonds. The [Condition] is met.

Case 2: $G = R$. In this special case, the balance of the issued bonds does not change. The ACB can realize the expected total coin supply (whose annual growth rate is G) by redeeming bonds on their redemption dates (whose interest rate is R). The [Condition] is met.

Case 3: $G < R$. In this case, the balance of the issued bonds increases. The ACB needs to issue bonds faster than redeeming bonds because the expected annual growth of the total coin supply (G) is smaller than the annual interest rate of the bonds (R). If the balance of the issued bonds is likely to go to infinity, users will stop buying the bonds and thus the [Condition] breaks. However, $G < R$ does not break the [Condition] immediately. As demonstrated in Japan, the United States and other advanced countries, private sectors can tolerate some degree of the increased balance of the government's debt [22]. Where is the threshold?

The key observation is that there is a limit to people's abilities and willingness about how much they can buy bonds. If people believe that other people will have abilities and willingness to buy bonds, people believe that the ACB has the ability of adjusting the exchange rate toward 1.0. Then people buy bonds and thus the [Condition] is met (regardless of the balance of the issued bonds). This infinite incentive recursion plays a critical role here. Money is accepted as money by everybody merely because it is accepted as money by everybody else [7, 8]. Similarly, bonds are accepted as bonds by everybody merely because they are accepted as bonds by everybody else. If the gap between G and R is small, $G < R$ will not break the infinite incentive recursion and not break the [Condition]. As [MMT-II] states, money creation and the increased balance of the debt are not a problem as long as they do not lead to inflation (i.e., as long as they do not break the infinite incentive recursion).

The conclusion is that **the [Condition] is met if $G >= R$, or $G < R$ but the gap is limited**. As long as the ACB sets a lower value to R (e.g., 1%), the [Condition] is likely to be met.[8]

[8] If G is consistently lower than 1%, it means that the economy is not growing. This is a far more fundamental problem before worrying about the incentive recursion.

6 Implementation

6.1 Smart contracts

JohnLawCoin is implemented as a smart contract on Ethereum.[9] The smart contract is implemented to meet the following requirements:

- There is truly no gatekeeper. The ACB is fully automated and no one (including the author of the smart contract) has privileges to influence the ACB. This can be verified by the fact that the smart contract has no operations that need permissions.[10]
- The smart contract is self-contained. There are no dependencies on other smart contracts and external services.
- All operations are guaranteed to terminate in the time complexity of O(1). The time complexity of each operation is determined solely by the input size of the operation and not affected by the state of the smart contract.

6.2 Coins and bonds

The coins are implemented as ERC20 tokens [23]. There are only three ways to get the coins; 1) ask someone to transfer coins to your account (e.g., by purchasing JohnLawCoin at an external currency exchanger), 2) contribute to the oracle and get the reward and 3) get an interest rate by holding bonds. It is worth noting that the ACB does not have the ability to exchange ETH with the coins. This is an intentional design because defining the conversion rate between fluctuating ETH and JohnLawCoin contradicts the goal of stabilizing the JohnLawCoin / USD exchange rate.

The bonds are implemented as non-transferable tokens. Bondholders can buy and redeem the bonds at the ACB but cannot transfer the bonds to others.[11][12]

6.3 Constant values

Table 2 shows the quantized exchange rates supported by the oracle.[13] The bond issue price B_{issue} depends on the exchange rates to reflect the risks. The bonds are designed as zero-coupon bonds, and R, T (measured in days), B_{redemp} and B_{issue} meet $B_{\text{issue}}(1+R)^{365/T} = B_{\text{redemp}}$. These values are carefully chosen based on the author's simulation results. The simulation tested how the total coin supply, the bond supply and the exchange rate changes over time in the next 20 years using various parameters.

Table 3 shows other constant values.[14] The initial coin supply is minted in the genesis account (created by the author of the smart contract). It is important to give a substantial amount of coins to the genesis account so that the genesis account can have power to determine the exchange rate until the ecosystem stabilizes. When real-world currency exchangers appear and the oracle gets a sufficient number of honest voters to agree on the real-world exchange rate consistently, the genesis account can lose its power by decreasing the coin balance, moving the oracle to a fully decentralized system.[15] This mechanism is mandatory to bootstrap the ecosystem and stabilize the exchange rate successfully.

[9] The implementation is available on GitHub (https://github.com/xharaken/john-law-coin).

[10] The only exceptions are a few operations like upgrading / pausing / unpausing the smart contract, which is needed to fix bugs in a development phase.

[11] If the bonds are transferable, that produces a strange situation. If the bonds are transferable, no users will hold the coins because they cannot get any interest by holding the coins whereas they can get an interest by holding the bonds. For the bonds to work as a mechanism to adjust the total coin supply, their liquidity needs to be restricted.

[12] This is only saying that the bonds are not transferable at the layer of the ACB. An open market to exchange bonds may emerge.

[13] You may wonder why bonds are issued when the exchange rate is 1.0 or above. In the ACB algorithm, it's possible that *acb.bond_budget* becomes positive when the exchange rate is 1.0 or above if the bond redemption happens too much. In this case, the ACB issues new bonds until *acb.bond_budget* goes down to zero.

[14] With these values, the maximum increase of the total coin supply per week is $k(1.4 - 1.0) = 4\%$.

[15] Until a real-world currency exchanger appears, the genesis account will keep voting for the exchange rate of

Table 2: The quantized exchange rates supported by the oracle.

E	B_{issue}	B_{redemp}	T	R
1 coin = 0.6 USD	950 coins	1000 coins	12 weeks	25.0%
1 coin = 0.7 USD	965 coins	1000 coins	12 weeks	16.7%
1 coin = 0.8 USD	978 coins	1000 coins	12 weeks	10.1%
1 coin = 0.9 USD	990 coins	1000 coins	12 weeks	4.46%
1 coin = 1.0 USD	997 coins	1000 coins	12 weeks	1.31%
1 coin = 1.1 USD	997 coins	1000 coins	12 weeks	1.31%
1 coin = 1.2 USD	997 coins	1000 coins	12 weeks	1.31%
1 coin = 1.3 USD	997 coins	1000 coins	12 weeks	1.31%
1 coin = 1.4 USD	997 coins	1000 coins	12 weeks	1.31%

6.4 APIs

In contrast with all the complex mechanisms explained so far, the APIs exposed by the ACB contract are simple and easy to understand. The ACB contract exposes only three APIs:

function vote(*committed_hash*, *revealed_exchange_rate*, *revealed_salt*)
The message sender votes on the oracle. With this function, the message sender can commit a vote to the current phase, reveal their vote in the prior phase, and reclaim their deposited coins and get a reward for their vote in the next prior phase at the same time. *committed_hash* is the hash committed to the current phase. *revleaed_exchange_rate* and *revealed_salt* reveal the quantized exchange rate and the salt for their vote in the prior phase. The reclaimed coins and the reward in the next prior phase are sent to the message sender's coin balance. The function returns a tuple of three values: 1) whether the commit succeeded or not, 2) whether the reveal succeeded or not and 3) the amount of the reclaimed coins and the reward. One message sender can make one vote per period (i.e., 1 week).

function purchaseBonds(*count*)
The message sender purchases *count* bonds from the ACB. The function returns the redemption timestamp of the purchased bonds (i.e., the current timestamp $+T$).

function redeemBonds(*redemption_timestamps*)
The message sender redeems bonds. *redemption_timestamps* is a list of redemption timestamps that specify the bonds. The function returns the number of bonds that are successfully redeemed.

As a result of calling these APIs, the message sender's coin balance and bond balance may change. The Coin contract exposes ERC20 token APIs. The message sender can query their coin balance and the total coin supply, and transfer their coins to others. The Bond contract only exposes APIs to query their bond balance and the total bond supply because bonds are not transferable.

1.1 and other voters are expected to follow. This means that the total coin supply will increase gradually in the bootstrap phase. Once a real-world currency exchanger appears, the genesis account will use the real-world exchange rate and other voters are expected to follow. When the oracle gets a sufficient number of honest voters to agree on the real-world exchange rate, the genesis account will decrease the coin balance and lose the power.

Table 3: Constant values.

D (A voter needs to deposit D% of their coin balance to the oracle.)	10%
C (The oracle evenly distributes C% of the reward to the voters who voted for the "truth" exchange rate. The rest of the reward is distributed to the voters in proportion to the coins they deposited.)	10%
Voters who voted for the "truth" exchange rate or its surroundings can reclaim the coins they deposited. What is the threshold?	0.1 (For example, if the "truth" exchange rate is 1.1, voters who voted for 1.0, 1.1 and 1.2 can reclaim the coins they deposited)
The ACB obtains the exchange rate from the oracle and adjusts the total coin supply every period. What is the duration of the period?	1 week
k (A damping factor to avoid minting or burning too many coins in one period.)	10%
Initial coin supply	2100000 coins

7 Conclusions

JohnLawCoin is a stablecoin realized by an Algorithmic Central Bank. The system is fully decentralized and there is truly no gatekeeper. No gatekeeper means there is no entity to be regulated.

JohnLawCoin is a real-world experiment to verify the assumption that there is a way to establish the infinite incentive recursion and stabilize the currency price without holding any collateral. Some of the monetary protocols are inspired by MMT. If JohnLawCoin is successful and proves the assumption is correct, it will provide interesting insights for both non-fiat currencies and fiat currencies; i.e., 1) there is a way for non-fiat cryptocurrencies to stabilize their currency price without having any gatekeeper, and 2) there is a way for central banks of developing countries to implement a fixed exchange rate system without holding adequate USD reserves. This will upgrade human's understanding about money.

If you are interested in JohnLawCoin, please contribute to the oracle. Early adopters can get more coins. The more users the ecosystem gets, the more likely real-world currency exchangers for JohnLawCoin appear and thus the ecosystem bootstraps. If you have any questions, please add comments to the GitHub issues at https://github.com/xharaken/john-law-coin.

About the name of "JohnLawCoin": In the early 18th century, John Law [1, 2, 8] invented a brand-new method to run a central bank with fiat currencies, which shifted the currency system in France from metalism to chartalism. In the end, Law's system failed and triggered an economic crisis in France. However, he was right. His system upgraded human's understanding about money and established the foundation of modern fiat currencies.

References

[1] Glyn Davies. *History of money*. University of Wales Press, 2010.

[2] Niall Ferguson. *The Ascent of Money: A Financial History of the World*. Penguin, 2008.

[3] Tether: Fiat currencies on the bitcoin blockchain. Technical report, Tether, 2016.

[4] Libra whitepaper. Technical report, Libra Association, https://wp.diem.com/en-US/wp-content/uploads/sites/23/2020/04/Libra_WhitePaperV2_April2020.pdf, 2020.

[5] The maker protocol: Makerdao's multi-collateral dai (mcd) system. Technical report, Maker Foundation, 2020.

[6] Samuel Brooks, Anton Jurisevic, Michael Spain, and Kain Warwick. Haven: A decentralised payment network and stablecoin. Technical report, Synthetix, 2018.

[7] Katsuhito Iwai. The boostrap theory of money: A search-theoretic foundation of monetary economics. *Structural Change and Economic Dynamics*, 7(4):451–477, 1996.

[8] Katsuhito Iwai. Evolution of money. *Evolution of Economic Diversity*, pages 396–431, 1997.

[9] Stephanie Kelton. *The Deficit Myth: Modern Monetary Theory and the Birth of the People's Economy*. Hachette UK, 2020.

[10] L. Randall Wray. *Modern Money Theory: A Primer on Macroeconomics for Sovereign Monetary Systems*. Palgrave Macmillan, 2016.

[11] Thomas I Palley. Money, fiscal policy, and interest rates: A critique of modern monetary theory. *Review of Political Economy*, 27(1):1–23, 2015.

[12] Nader Al-Naji, Josh Chen, and Lawrence Diao. Basis: A price-stable cryptocurrency with an algorithmic central bank. Technical report, BASIS, 2018.

[13] Robert Sams. A note on cryptocurrency stabilisation: Seigniorage shares. Technical report, 2015.

[14] Eiland Glover and John W Reitano. The kowala protocol: A family of distributed, self-regulating, asset-tracking cryptocurrencies. Technical report, Kowala, 2018.

[15] Steve Ellis, Ari Juels, and Sergey Nazarov. Chainlink: A decentralized oracle network. Technical report, Chainlink, 2017.

[16] John Adler, Ryan Berryhill, Andreas Veneris, Zissis Poulos, Neil Veira, and Anastasia Kastania. Astraea: A decentralized blockchain oracle. In *2018 IEEE international conference on internet of things (IThings)*, pages 1145–1152. IEEE, 2018.

[17] Maximilian Wöhrer and Uwe Zdun. Design patterns for smart contracts in the ethereum ecosystem. In *2018 IEEE International Conference on Internet of Things (iThings)*, pages 1513–1520. IEEE, 2018.

[18] Vitalik Buterin. Schellingcoin: A minimal-trust universal data feed. Technical report, https://blog.ethereum.org/2014/03/28/schellingcoin-a-minimal-trust-universal-data-feed/, 2014.

[19] Pingyang Gao. Keynesian beauty contest, accounting disclosure, and market efficiency. *Journal of Accounting Research*, 46(4):785–807, 2008.

[20] N. Gregory Mankiw. *Macroeconomics*. Worth, 2019.

[21] Evsey D Domar. The "burden of the debt" and the national income. *The American Economic Review*, 34(4):798–827, 1944.

[22] Douglas W Elmendorf and N Gregory Mankiw. Government debt. *Handbook of macroeconomics*, 1:1615–1669, 1999.

[23] Fabian Vogelsteller and Vitalik Buterin. Eip-20: Erc-20 token standard, 2015.