Research Article

Payment Guaranteed Polynomial Exchange Rate Scheme and Its Application to Cryptocurrency Swaps

Youngseo Yoo[*]**, Dongil Seo and Minhyun Kim**

Common Computer, 17F 1769, 382, Gangnam-daero, Gangnam-gu, Seoul, Korea Republic
lia@comcom.ai; seo@comcom.ai; kimminhyun@comcom.ai
*Correspondence: lia@comcom.ai

Abstract: In this paper, we introduce a scheme, called Polynomial Exchange Rate Scheme (PERS), to generate exchange rate functions for token swap systems, and show that the functions generated are consistent, stable, and resilient. We show that payments are guaranteed in PERS if the Single Circulation Source principle is adopted (i.e., PG-PERS). Compared to the existing deposit-based exchange rate schemes, PG-PERS is a scheme that requires no initial key token deposit and its price changes have relatively stable rates especially in extreme cases. As an application of PG-PERS, we present a token swap service, called Fanco Swap, for swapping the ERC20 token used on aFan, an incentivized social media platform, and Ether coin. We also cover several practical issues such as precision and computation cost problems and the solutions to them, which adopted in the implementation of Fanco Swap. The paper contains a comprehensive survey on existing cryptocurrency exchange services and their pricing mechanism, followed by a formal development of the proposed exchange rate scheme and its comparison with one of the most representative existing exchange rate schemes.

Keywords: *Token Swap System; Exchange Rate Scheme; Cryptocurrency; Smart Contract; ERC20 Token*

1. Introduction

Since the advent of Ethereum blockchain and smart contracts[1], more than 230,000 tokens[1] have been created on Ethereum Network and are being used in numerous areas; however, only a very small fraction of those tokens are listed on the small set of popular cryptocurrency exchanges.[2] Many exchanges are highly selective in approving which tokens to be listed on their systems, and it often takes a lot of time, effort and money to meet the exchanges' requirements. Furthermore, even when a project succeeds in listing its token on an exchange, without a substantial amount of users or trading volume, it tends to suffer from price volatility. This is backed by Hamrick et al.'s recent research on the profitability of pumping, artificial inflating of the price[2]. Their analysis shows that pumping

---

[1] https://etherscan.io/tokens.
[2] See tokens with "Ethereum" Platform at https://coinmarketcap.com/tokens/views/all/.

obscure coins with low trading volume is much more profitable, i.e., small projects are more vulnerable to pump and dump schemes. Due to these reasons, listing on an exchange can be a double-edged sword for a certain type of cryptocurrency or token; this generates a need for a system where developer communities and project owners can distribute tokens for their projects in a relatively predictable and stable manner and where investors can take part in early-stage projects.

With the increasing demand, there has been active research on such exchange systems that enable direct swap of cryptocurrencies without order books. Two issues need to be addressed in order to develop a cryptocurrency or token swap service that doesn't rely on an order book. First, the service has to guarantee atomic transfers of assets. That is, the state of a transaction can be either entirely completed or entirely undone. Secondly, the service should offer reasonable exchange rates for the participants. Various solutions to these issues have been proposed which can be categorized into two types of approaches, with both focusing on the atomicity of transactions. One of the approaches is to design an Atomic Swap protocol that enables wallet-to-wallet transfers. The protocol's complete form was first presented by Tier Nolan in 2013[3], and many projects have adopted it since then[4]. Another approach utilizes smart contracts to guarantee atomicity and incorporates well-designed exchange rate functions to generate exchange rates programmatically. Uniswap[3] and Bancor[5] are well-known examples of these efforts. On Uniswap, anyone can list an ERC20 token and create an exchange for swapping the token with Ether coin (symbol: *ETH*). Bancor has its own token (symbol: *BNT*) that acts as an intermediary of any two cryptocurrencies that a user wishes to swap. Many of the new exchange systems determine the exchange rates using functions of the reserve fund or the total supply of tokens. In this paper, we propose an exchange rate scheme that determines exchange rates with polynomial functions of the total balance of tokens, and discuss and present solutions to some of the problems that arise when implementing the scheme in practice.

2. Related Works

Numerous trading platforms have existed long before cryptocurrencies, and when cryptocurrencies became popular, people started developing exchanges that worked in a similar way to the traditional trading platforms. These platforms are often called "centralized exchanges" because they have centralized servers that control user accounts, order books, and matching of orders. Generally, these centralized exchanges have high liquidities and trading volumes, and are able to process orders quickly. Coinbase[4], Binance[5], and Bittrex[6] are some of the widely known centralized cryptocurrency exchanges. Unfortunately, the centralized nature of these exchanges poses a single point of failure, and as of December 2019, it is reported that more than $1.74 billion *USD* has been stolen over a dozen different exchanges[7]. Just like blockchains were developed to mitigate the vulnerabilities of centralized systems, decentralized exchanges (DEXs) have been gaining popularity. DEXs operate peer-to-peer using smart contracts in such a way that no central entity holds the account secrets or funds of the users. Examples of DEXs include Oasis Trade[8] (formerly Eth2Dai), ShapeShift[9], and Bisq[10].

Many DEXs essentially migrate the order books and matching algorithms to smart contracts, resulting in slower processing of orders and lower trading volumes. Exchanges have since adopted hybrid methods where a portion of the system runs off-chain for performance and the rest remains on-chain for security. IDEX[11], EtherDelta[12] and 0x Protocol[13] are a few of the hybrid DEXs. One of the

---

[3] https://uniswap.io

[4] https://www.coinbase.com/

[5] https://www.binance.com/en

[6] https://global.bittrex.com/

[7] https://www.ledger.com/academy/crypto/hacks-timeline/.

[8] https://oasis.app/trade/

[9] https://shapeshift.io/#/coins

[10] https://bisq.network/

[11] https://idex.market/eth/idex

[12] https://etherdelta.com/

[13] https://0x.org/

characteristics that the platforms mentioned until now have in common is that they keep order books, and the buying and selling orders affect the exchange rates.

Instead of relying on traditional order books, a few decentralized projects such as Bancor[5] and Uniswap take a different approach by adopting price making algorithms. Such price making algorithms are referred to as automated market makers (AMMs), even though they behave slightly differently than traditional AMMs in prediction markets. DEXs like Uniswap and Bancor aim to always provide liquidity through smart contracts that calculate buy and sell prices.

In Bancor's system, token price is a function of the balance of the connector token, outstanding supply of the token, and the connector weight. Their price equation is $price = \frac{R}{S \times F}$, where R = connector balance (reserve), S = token's outstanding supply, and F = connector weight (constant fractional reserve ratio)[6]. Driven from the equation, the amount T of token a buyer would get by paying E amount of connector token is $T = S_0((1 + \frac{E}{R_0})^F - 1)$. If the buyer specifies the buy amount T and wants to calculate how much connector token s/he needs to pay, the formula they propose is: $E = R_0(\sqrt[F]{1 + \frac{T}{S_0}} - 1)$. Although Bancor has its own advantages, their connector token system, the formulae, and their implementation can be over-complicated for freshly starting projects. Furthermore, the project has been criticized for its error-prone arithmetic functions[7].

On the other hand, Uniswap employs a simpler function for its system, namely the Constant Product Market Maker model[8]. Given two cryptocurrencies X and Y with balances x and y, respectively, the Constant Product Market Maker model has an invariant value k that should equal the product of x and y at any time, i.e., $x \cdot y = k$. The price of X with respect to Y, which is $\frac{dy}{dx}$, depends on the amount of cryptocurrency reserved in the exchange. In fact, when a user wants to buy dx amount of X when the exchange's balances of X and Y are x_t and y_t, respectively, the amount dy of Y that user needs to pay can be expressed with just dx, x_t and k. To satisfy the invariance after the swap, $(x_t - dx)(y_t + dy) = k$. Therefore, $dy = \frac{k}{x_t - dx} - y_t = \frac{k}{x_t - dx} - \frac{k}{x_t}$. Although Uniswap's system is easy to understand and simple enough to be implemented as a smart contract, it has a couple of caveats. It requires the initial values of x and y to be greater than zero, and in order to satisfy the target initial price of a token, obtain a reasonable rate of change of the price, as well as have enough liquidity, a very large amount of Ether coin is needed in the beginning. Also, the deposit-based exchange rate schemes show dramatic price changes when the balance of the deposited tokens in the exchange system approaches its limits (see Section 3.6 for more details). Thus the price making model proposed in this paper was conceived with the aforementioned restrictions in mind.

3. Polynomial Exchange Rate Scheme (PERS)

In this section, we provide a formal definition of the conditions the new exchange scheme should meet and introduce an exchange rate scheme that satisfies those conditions. We also introduce the Single Circulation Source principle and how it can be applied to a token swap system to guarantee payments for the circulated derived token units. Finally, we compare the proposed scheme with one of the most representative deposit-based exchange rate schemes.

3.1 Terminology

- *Exchange system*: A system where two tokens (a key token and a derived token) can be swapped with each other.
- *Key token T_k*: A widely used token e.g. Ether coin
- *Derived token T_d*: A newly issued token e.g. ERC20 token
- *Exchange rate function*: A function, denoted by R, that defines the exchange rate of T_d with respect to T_k, parameterized by the current states of the exchange system. The exchange rate function for T_k in T_d is denoted by r which equals $\frac{1}{R}$. For example, if 1 unit of T_k is exchanged for 10,000 units of T_d, $R = 0.0001$ and $r = 10,000$.
- *Initial exchange rate*: R_0 and r_0 are the initial values of R and r, respectively.

- *Exchange rate scheme*: A set of exchange rate functions with different parameters.
- *Resilience pressure*: The pressure pushing the exchange rate to the initial value, which is given by the gradient of the exchange rate function.
- *Token circulation*: The amount of a token circulated through the market, which is different from the total supply of the token.

3.2 Problem Definition

At a certain timestamp, let S be the total supply of the derived token from all the swaps so far in $T_d \rightarrow T_k$ direction and let D be the total demand for the derived token from all the swaps so far in $T_k \rightarrow T_d$ direction. Then the *total balance B* of the derived token is defined as $B = S - D$. Similarly, the *total balance b* of the key token is defined as $b = s - d$ where s and d are the total supply of and the total demand for the key token.

Total balances B and b are key parameters of the exchange rate (an extreme case here is when the total balance goes to either $-\infty$ or ∞, which means a collapse of the exchange system) so the exchange rate functions R and r can be defined as functions of B and b, respectively. The next parameter is the resilience pressure p. Larger p values mean higher resilience pressure on the exchange rate to return to the initial values R_0 and r_0. The final parameters are the initial exchange rates R_0 and r_0.

The problem we are addressing in this paper is to find an exchange rate scheme parameterized by B, p, and R_0 (or b, p, and r_0) that meets the following conditions:

- Consistent: When tokens are swapped in a round trip without any external interference (for example, a derived token unit is swapped to the key token and then back to the derived token), the swaps are consistent if and only if the amounts of the tokens after the swaps are equal to their amounts before the swaps.
- Stable: If the total balance B (or b) returns to its initial value 0, the exchange system returns to its initial state.
- Resilient: An exchange in one direction should increase the pressure for the exchange in the opposite direction, meaning an increase or decrease of the exchange rate to return to the initial state. For example, an exchange in $T_k \rightarrow T_d$ direction should increase the exchange rate R (or decrease r), i.e., makes the price of T_d in T_k higher. On the contrary, an exchange in $T_d \rightarrow T_k$ direction should decrease the exchange rate R (or increase r), i.e., lowers the price of T_d in T_k.

We refer to these conditions as the *CSR requirements* for convenience in this paper.

3.3 Proposed Scheme

Our strategy is to define exchange rates as a polynomial function of total balances.

3.3.1 When $B \leq 0$ (i.e., $b \geq 0$)

Let us first consider the exchange rate function for *negative B* values, which is the case when the derived token has been sold more than it's been bought from the exchange's point of view. Let X be the absolute value of B, i.e., $X = -B$, and Y be the absolute value of b, i.e., $Y = b$. The exchange rate function $R(X)$ is defined as

$$R(X) = \frac{dY}{dX} = \alpha X^p + R_0 \text{ for } X \geq 0 \tag{1}$$

where $p > 0$ is the resilience pressure and $\alpha > 0$ is a constant, and R_0 is the initial exchange rate of T_d in T_k. Figure 1 shows examples of polynomial exchange rate functions $R(X)$ generated from (1).
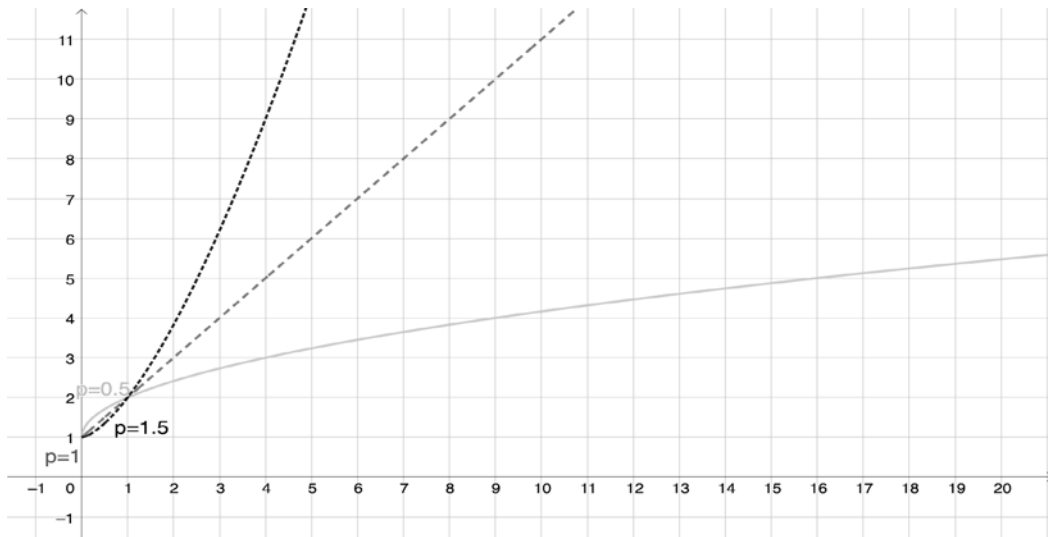
Figure 1. Polynomial exchange rate functions $R(X)$ for $X = -B \geq 0$ with different resilience pressure values p = 0.5, 1, and 1.5, and $R_0 = 1$.

Theorem 1: $R(X)$ *satisfies the CSR Requirements for $X \geq 0$ if exchanges are made in a sequence.*

Proof. For an exchange in $T_k \to T_d$ direction, $R(X)$ satisfies the *resilient* condition of the CSR Requirements as $R(X_t + \Delta X) > R(X_t)$ for $X_t, \Delta X > 0$ where X_t and $X_t + \Delta X$ are the absolute values of the total balance B of T_d before and after the exchange, respectively. It is trivial that the condition is satisfied for an exchange in the opposite direction $T_d \to T_k$.

The *consistent* condition of the CSR Requirements is satisfied by using the integral of $R(X)$ over $[X_t, X_t + \Delta X]$ as the exchange rate where X_t is the value of X before an exchange and ΔX is the amount of the derived token to be exchanged. Now the integral of $R(X)$ is

$$\int R(X) = \frac{\alpha}{p+1} X^{p+1} + R_0 X + C$$

where C is an integral constant. So the amount of the key token ΔY exchanged for ΔX amount of the derived token is

$$\Delta Y = \int_{X=X_t}^{X_t+\Delta X} R(X) = \frac{\alpha}{p+1} ((X_t + \Delta X)^{p+1} - X_t^{p+1}) + R_0 \Delta X. \tag{2}$$

Finally, it can be shown that the exchange system satisfies the stable condition of the CSR Requirements as follows. The exchange system consists of three variables, X, Y, and R:

1. By $R(0) = R_0$, R returns to the initial value if the total balance B returns to its initial value 0.
2. Consider two cases where one consists of two sequential exchanges with amounts ΔX_1 and ΔX_2, and another consists of an exchange with amount $\Delta X_3 = \Delta X_1 + \Delta X_2$. By (2),

$$\Delta Y_3 = \int_{X=X_t}^{X_t+\Delta X_3} R(X) = \int_{X=X_t}^{X_t+\Delta X_1} R(X) + \int_{X=X_t+\Delta X_1}^{X_t+\Delta X_1+\Delta X_2} R(X) = \Delta Y_1 + \Delta Y_2$$

where ΔY_1 and ΔY_2 are the amounts of the key token exchanged for the derived token amounts ΔX_1 and ΔX_2 in the two sequential exchanges, respectively. So, the two cases are equivalent, which means the exchange operation is *transient*. Thus, a series of exchanges resulting in the final total balance B of value 0 can be collapsed into one exchange with $\Delta X = 0$ and in this case $\Delta Y = 0$ as well. So, if variable X returns to its initial value, variable Y also returns to its initial value.
Q.E.D.

3.3.2 When $B \geq 0$ (i.e., $b \leq 0$)

For positive B, we apply the same functions to b as we did for B. Let x be the absolute value of b, i.e. $x = -b$ and y be the absolute value of B, i.e., $y = B$, then the exchange rate function $r(x)$ is defined as

$$r(x) = \frac{dy}{dx} = \beta x^p + r_0 \text{ for } x \geq 0 \tag{3}$$

where $p > 0$ is the resilience pressure, $\beta > 0$ is a constant, and r_0 is the initial exchange rate of T_k in T_d, i.e., $r_0 = \frac{1}{R_0}$.

Theorem 2: $r(x)$ *satisfies the CSR Requirements for* $x \geq 0$.

Theorem 2 can be proven in the *same* way as Theorem 1.

3.4 Case Study: Linear Function (i.e., $p = 1$)

In this section, we provide an in-depth study of the exchange rate scheme for a special case of $p = 1$.

3.4.1 When $B \leq 0$ (i.e., $b \geq 0$)

From (1),

$$R(X) = \frac{dY}{dX} = \alpha X + R_0 \tag{4}$$

where X is the absolute value of B, i.e. $X = -B$ and Y is the absolute value of b, i.e., $Y = b$, and R_0 is the initial exchange rate of T_d in T_k. Figure 2 shows examples of linear exchange rate functions $R(X)$ generated from (4).
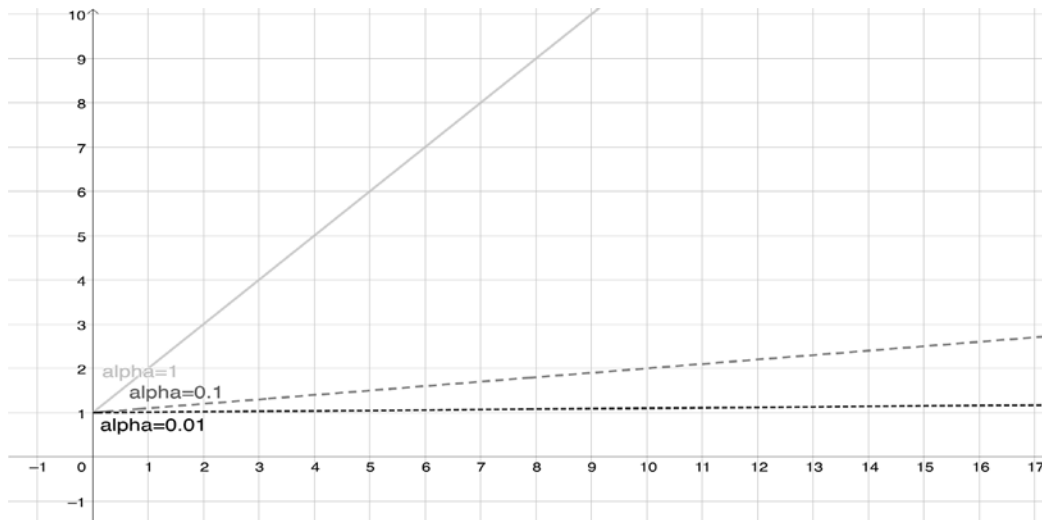

Figure 2. Linear exchange rate functions $R(X)$ for $X = -B \geq 0$ with different alpha values =1, 0.1, and 0.01, and R_0 = 1.

So, $Y = \int (\alpha X + R_0) dX = \frac{\alpha}{2} X^2 + R_0 X + C$, where C is the integral constant.

From $Y = 0$ for $X = 0$,

$$Y = \frac{\alpha}{2} X^2 + R_0 X. \tag{5}$$

Figure 3 shows the relationship between the total balance $Y = b$ of the key token and the total balance $X = -B$ of the derived token from (5).

From (5), the amount ΔY of the key token T_k exchanged for ΔX amount of the derived token T_d when $X = X_t$ is derived as

$$\Delta Y = \frac{\alpha}{2}((X_t + \Delta X)^2 - X_t^2) + R_0 \Delta X. \tag{6}$$

By solving the quadratic equation (5), we get

$$X = \frac{1}{\alpha}(\sqrt{2\alpha Y + R_0^2} - R_0). \tag{7}$$

So the amount ΔX of the derived token T_d exchanged for ΔY amount of the key token T_k when $Y = Y_t$ is derived as

$$\Delta X = \frac{1}{\alpha}(\sqrt{2\alpha(Y_t + \Delta Y) + R_0^2} - R_0) - X_t. \tag{8}$$

So, using (6) and (8), we can get the amount of tokens exchanged given ΔX and ΔY, respectively. If we apply (7) to (4), we get the following:

$$R(Y) = \sqrt{2\alpha Y + R_0^2}. \tag{9}$$

Figure 4 shows examples exchange rate functions $R(Y)$ generated from (9).

From (9),

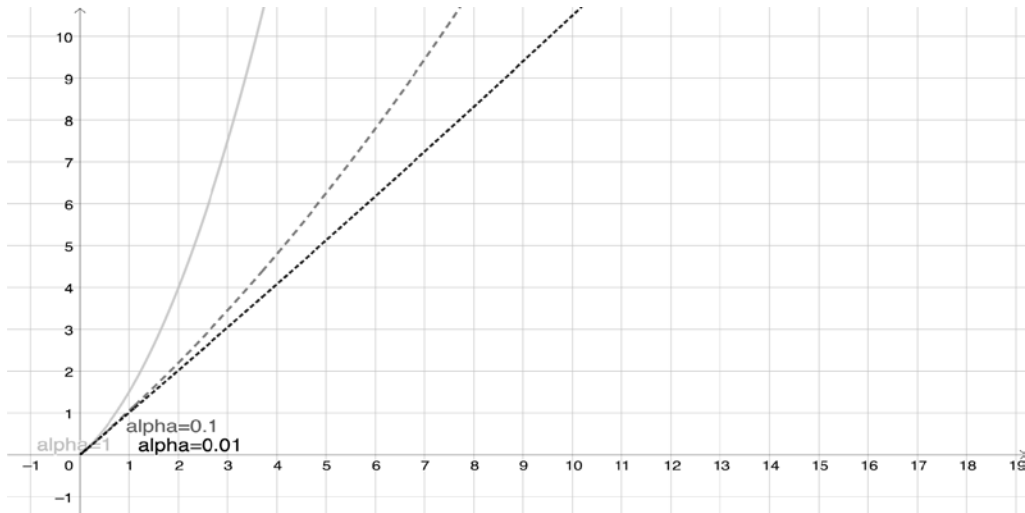$$\alpha = \frac{1}{2Y}(R^2(Y) - R_0^2). \tag{10}$$

Figure 3. The total balance $Y = b$ of the key token for the total balance $X = -B \geq 0$ of the derived token with different alpha values α =1, 0.1, and 0.01, and R_0=1.

Using this formula, we can get α for a condition given by the relation between Y_k and $R(Y_k)$. For example, if the condition is given as $R(Y_k) = kR_0$,

$$\alpha = \frac{(k^2-1)R_0^2}{2Y_k}. \tag{11}$$

So, if $k = 10$ for $Y_k = 33,333\ ETH$, i.e., $10^7\ USD$ (assuming $1\ ETH = 3 \times 10^2\ USD$) and $R_0 = 0.0001$, we get $\alpha = 14,850,000 \times 10^{-18}$, i.e., $14,850,000\ wei$.
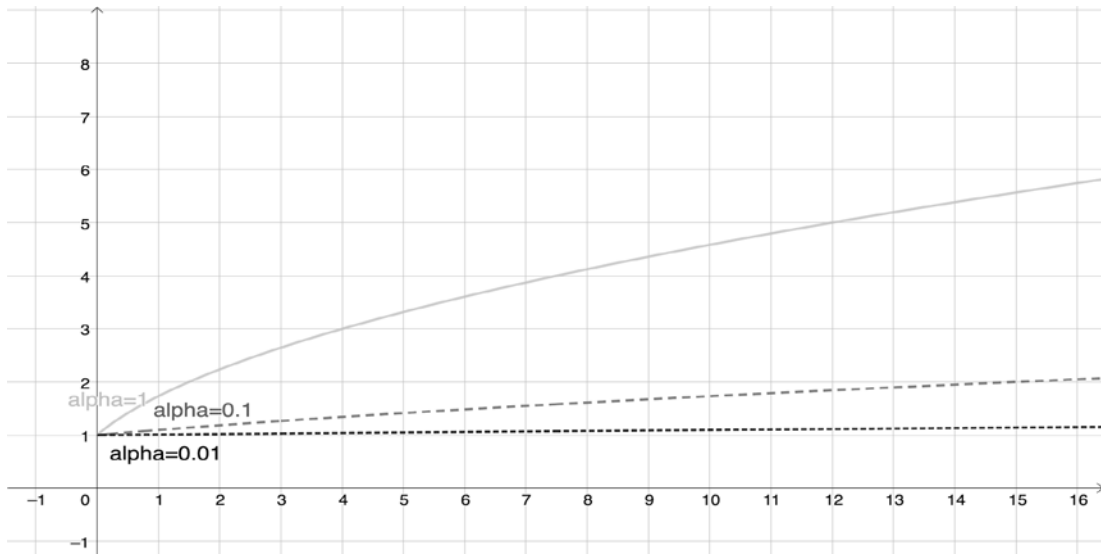

Figure 4. Exchange rate functions $R(Y)$ for $Y = b \geq 0$ with different alpha values α =1, 0.1, and 0.01, and $R_0 = 1$.

Instead of applying a predefined R_0 value, we can derive R_0 and α values by applying some fundamental numbers like the starting total balance $B_s\ (= -X_s)$ of the derived token and the key token budget $b_s\ (= Y_s)$, which is effectively having a constraint that the (X, Y) graph should pass a (X_s, Y_s) point. From the equations (5) and (11), following equations are derived for R_0 and α, respectively:

$$R_0 = \frac{-2Y_k+2\sqrt{Y_k^2+(k^2-1)Y_kY_s}}{(k^2-1)X_s}. \tag{12}$$

$$\alpha = \frac{4Y_k-4\sqrt{Y_k^2+(k^2-1)Y_kY_s}+2(k^2-1)Y_s}{(k^2-1)X_s^2}. \tag{13}$$

If we apply $k = 10$, $X_s = 111135874.936329$, $Y_s = 1\ ETH$, and $Y_t = 33,333\ ETH$ to (11) and (12), for example, we get $R_0 = 0.0002000159754300310 \times 10^{-18} = 20001597543003\ wei$ and $\alpha = 594095\ wei$.

From (9), the volatility $\frac{dR}{dY}$ of the exchange rate R with respect to the key token is

$$\frac{dR}{dY} = \frac{\alpha}{\sqrt{2\alpha Y + R_0{}^2}}. \tag{14}$$

From (5) and (14), we get

$$\frac{dR}{dY} = \frac{1}{X + \frac{R_0}{\alpha}} \tag{15}$$

which means

$$\frac{dY}{dR} = X + \frac{R_0}{\alpha} = -B + \frac{R_0}{\alpha}. \tag{16}$$

3.4.2 When $B \geq 0$ (i.e., $b \leq 0$)

From (3),

$$r(x) = \frac{dy}{dx} = \beta x + r_0 \tag{17}$$

where x is the absolute value of b, i.e. $x = -b$, and y is the absolute value of B, i.e., $y = B$, and r_0 is the initial exchange rate of T_k in T_d.

As in the case of $B \leq 0$, we can derive the following equations:

$$y = \frac{\beta}{2} x^2 + r_0 x. \tag{18}$$

$$\Delta y = \frac{\beta}{2}((x_t + \Delta x)^2 - x_t{}^2) + r_0 \Delta x\ . \tag{19}$$

$$x = \frac{1}{\beta}(\sqrt{2\beta y + r_0{}^2} - r_0). \tag{20}$$

$$\Delta x = \frac{1}{\beta}(\sqrt{2\beta(y_t + \Delta y) + r_0{}^2} - \sqrt{2\beta y_t + r_0{}^2}). \tag{21}$$

$$r(y) = \sqrt{2\beta y + r_0{}^2}. \tag{22}$$

If we apply $r(x_k) = kr_0$ to (17), we get

$$\beta = \frac{(k-1)r_0}{x_k}. \tag{23}$$

So, if $k = 10$ for $x_k = 33,333\ ETH$, i.e., $10^7\ USD$ (given $1\ ETH = 3 \times 10^2\ USD$) and $r_0 = 10000$, we get $\beta = 2.7 \times 10^{-18}$.

Similarly to the derivation of R_0 and α values from a fundamental coordinate (X_s, Y_s), r_0 and β can be derived by applying a fundamental coordinate (x_s, y_s) to (18) and (23):

$$r_0 = \frac{2x_k y_s}{2x_k x_s + (k-1)x_s{}^2}. \tag{24}$$

$$\beta = \frac{2(k-1)y_s}{2x_k x_s + (k-1)x_s{}^2}. \tag{25}$$

From (17), the volatility $\frac{dR}{dx}$ of the exchange rate R with respect to the key token is

$$\frac{dR}{dx} = \frac{d(\frac{1}{r})}{dx} = -\frac{\beta}{(\beta x + r_0)^2}. \tag{26}$$

From (20) and (26), we get

$$\frac{dR}{dx} = -\frac{1}{2y + \frac{r_0{}^2}{\beta}} \tag{27}$$

which means

$$\frac{dx}{dR} = -2y - \frac{r_0{}^2}{\beta} = -2B - \frac{r_0{}^2}{\beta}. \tag{28}$$

3.4.3 Combining the two cases: $B \leq 0$ (i.e., $b \geq 0$) and $B \geq 0$ (i.e., $b \leq 0$)

If we combine (4) and (22), we can get a graph of the exchange rate R for the total balance (which is X when $B \leq 0$ and y when $B \geq 0$) of the derived token as shown in Figure 5.
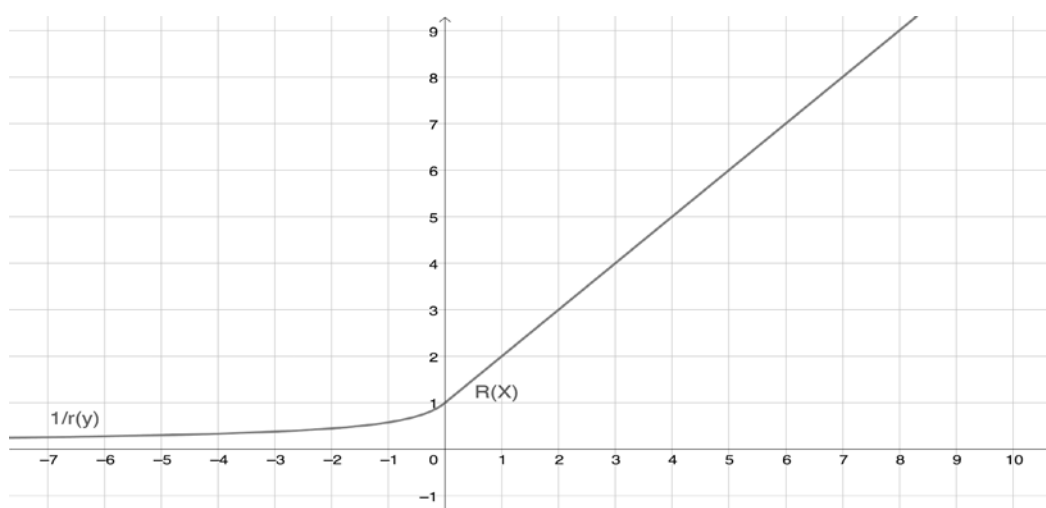
Figure 5. The exchange rate R for the total balance (which is X when $B \leq 0$ and y when $B \geq 0$) of the derived token. Drawn using (4) and (22) with $R_0 = \frac{1}{r_0} = 1$ and $\alpha = \beta = 1$.

Likewise, if we combine (9) and (17), we can get a graph of the exchange rate R for the total balance (which is Y when $b \geq 0$ and x when $b \leq 0$) of the key token as shown in Figure 6.
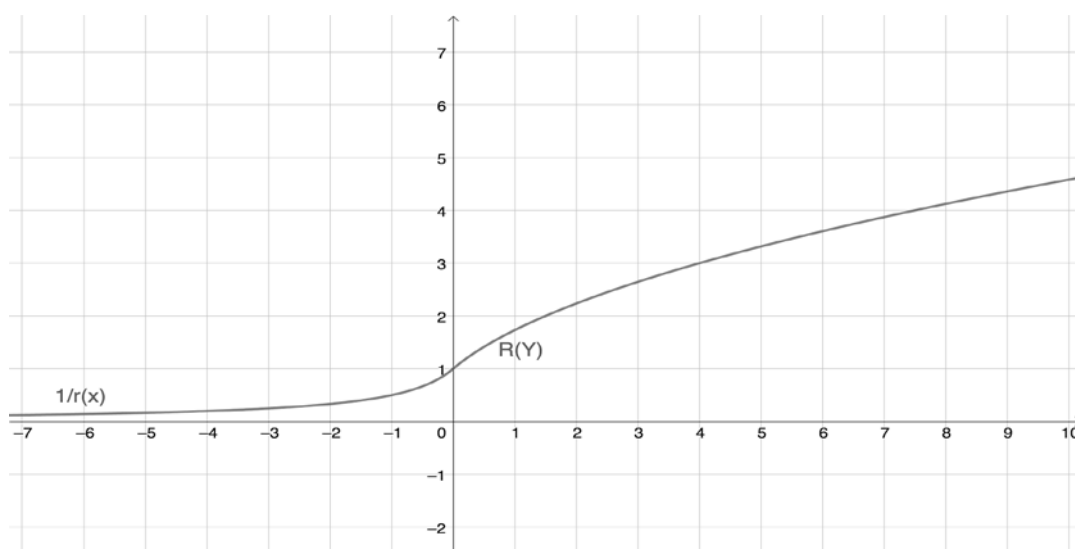

Figure 6. The exchange rate R for the total balance (which is Y when $b \geq 0$ and x when $b \leq 0$) of the key token. Drawn using (9) and (17) with $R_0 = \frac{1}{r_0} = 1$ and $\alpha = \beta = 1$.

3.5 Payment Guaranteed Exchange System

An exchange system can serve as a source of a derived token, directly affecting the amount of the token in circulation. For the system that is presented in this paper, we will adopt the following principle:

"The exchange system is the only circulation source of the derived token."

We call it the Single Circulation Source (SCS) principle, and it narrows down the possible cases to where $B \leq 0$ (i.e., $b \geq 0$), since no more than the amount of the derived token that has been swapped to the key token can be swapped back. In fact, the amount of the derived token in circulation always equals to $-B (= X)$.

This system has several advantages, which include but are not limited to: 1) *guaranteed payments* for the derived token in circulation, and 2) *transparency* in minting, burning, and pricing of the derived token. The payment guarantee is backed by the amount of reserved key token that has been swapped in exchange for the derived token, which in our scheme refers to b. Since the exchange

system is stable (by Theorem 1), any portion of the circulating derived token can be exchanged back to the key token at any time, although the amount of the paid key token depends on the exchange rate when the exchange is actually made. Moreover, when the amount of the derived token in circulation becomes zero, meaning the entire amount of the derived token in circulation returns to the exchange, the total key token paid is identical to the total key token originally used to circulate the derived token.

This system can withstand even the most extreme scenarios–one where $B \approx 0$ and another where $B \approx$ *derived token's total supply*. $B \approx 0$ means the exchange has just launched, or almost all the derived token that was bought by users have been sold back to the exchange. In that situation, the PG-PERS system doesn't break because there are no more derived tokens in circulation that can flow into the exchange. In the $B \approx$ *derived token's total supply* case on the other hand, the exchange has sold almost all of the minted derived token, so the derived token owners can mint more tokens to meet the market's demands.

We define such an exchange system a Payment Guaranteed exchange system and the exchange rate scheme built into the system a Payment Guaranteed Polynomial Exchange Rate Scheme, or PG-PERS. As X is the amount of the derived token in circulation in PG-PERS, (16) can be interpreted as follows:

> *"The amount of the key token needed to change the price of the derived token by a unit value is in linear proportion to the amount of the circulated derived token."*

So, the more the derived token is circulated, the stabler (i.e., less vulnerable) the token price becomes.

3.6 Comparison of PG-PERS and Uniswap

Table 1. A comparison of PG-PERS and Uniswap.

	PG-PERS		Uniswap (Constant-Product Market Maker)	
In-Bound Key Token	Derived Token Price	Price Increase (%p)	Derived Token Price	Price Increase (%p)
0	0.0000236294	0	0.0000236294	0
10	0.0000239781	1	0.0000236579	0
100	0.0000269137	14	0.0000239159	1
1,000	0.0000470993	99	0.0000265729	12
10,000	0.0001309899	454	0.0000608376	157
100,000	0.0004081156	1,627	0.0011730220	4,864
1,000,000	0.0012886265	5,353	0.0892486111	377,602
100,000,000	0.0128841204	54,426	8.6653856635	36,672,019

We compared the proposed scheme and Uniswap's Constant-Product Market Maker scheme in terms of the price change rate of the derived token. For realistic results, we used the same parameters used in Fanco Swap (see Section 4.1) for PG-PERS. For Uniswap, we tuned parameters so as to have the same initial price as PG-PERS (see Section 4.3.1). In order to set the initial prices identically, 16,541 units of the key token were needed for initial deposit in Uniswap's scheme, while none was needed for PG-PERS. Note that we assumed that there would be no exchange fees in both schemes. Table 1 shows the simulation results. As shown in the table, the rate of increase of the token price escalates dramatically in Uniswap compared to PG-PERS as the more derived token is bought by users, i.e., the amount of the key token flowing into the system increases.

4. Applications

4.1 Fanco Swap

Fanco Swap[14] is the first token swap service that utilizes the PG-PERS($p = 1$). Fanco (hereinafter, *FANCO*) is an ERC20 token that is currently used on aFan[15], which is an incentivized social media platform for creators and fans. Fanco Swap enables swaps between Fanco token and Ether coin, using equations (6) and (8) to calculate the exchanged amounts in swaps. For example, when a user pays *ΔY ETH* for some token, *ΔX FANCO* is computed using the equations and is given to the user. Figure 7 shows the high-level system structure of Fanco Swap. Its smart contract is written in Solidity[9], one of the most popular programming languages for Ethereum Virtual Machine (EVM)[10], and is deployed on the Ethereum Network.
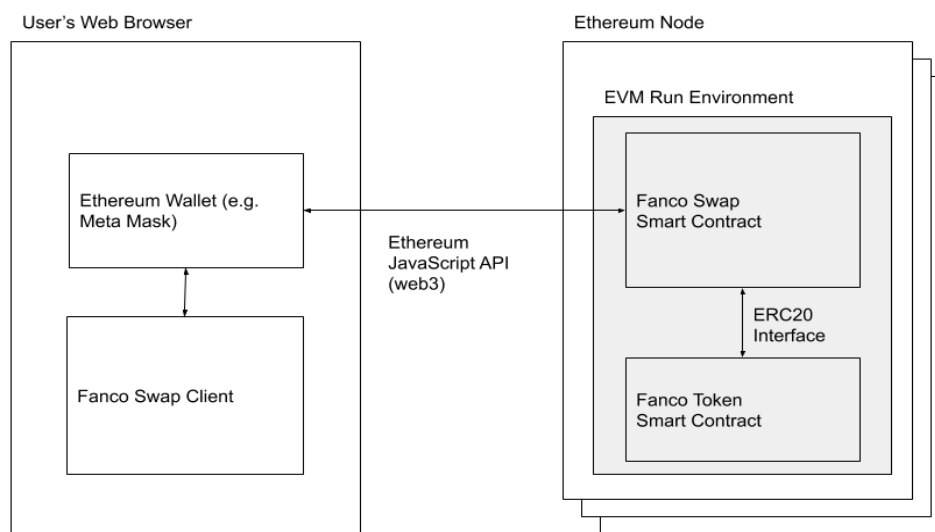

Figure 7. Fanco Swap System Structure

4.2 Precision Problem and Computation Cost Problem

The Fanco Swap's PERS algorithm implemented as a smart contract is executed in EVM run environment. Any computation done in a Blockchain setting needs to consider the asynchrony among Blockchain nodes, especially when it involves monetary values. Ethereum guarantees deterministic status changes by restricting the types of values EVM handles. For instance, EVM, as of writing, does not support floating-point numbers; instead, monetary values are recommended to be represented in the smallest unit called *wei*, which equals 10^{-18} of *ETH*. Another measure Ethereum takes to achieve deterministic execution and to prevent users from intentionally or unintentionally making nodes run their code forever is putting a price for each of the EVM operation[10]. A user who wishes to run code or process a transaction on Ethereum has to pay the costs (or "gas") upfront.

While these features of Ethereum help Ethereum's nodes to be in sync, they also complicate implementing PERS as an Ethereum smart contract. One of the problems we encountered while developing Fanco Swap was precision. As mentioned before, EVM does not support floating-point numbers nor more complex mathematical operations like square roots, which is required to solve for *ΔX* in equation (8).[16] A good technical solution is Newton's method for integer square roots that uses only addition and division operations[11]. However, the method has the quadratic convergence rate, and iterating and computing inside the for-loop on Ethereum costs extra gas fees that users would have to pay.

---

[14] https://afan.ai/swap

[15] https://afan.ai

[16] Note that equation (6), the equation for ΔY, is made up only of simple operations and can be solved on-chain.

As solutions to these precision and computation cost problems, we developed the "Off-chain Computation On-chain Verification (OCOV)" model and utilized the "Ratio Comparison" method. The OCOV model minimizes the smart contract computation load by delegating the heavy and complex computation to the off-chain client-side web page, leaving only the verification to the on-chain smart contract. In effect, computing the Ether amount ΔY user should get and the token amount ΔX to pay is done on the exchange web client with equation (8), and the ΔX and ΔY values are sent to the Ethereum network as a transaction. Then the verification of the ΔX and ΔY submitted by the transaction creator is made in EVM run environment. Also, the off-chain JavaScript code uses the big-number.js library[17] in order to handle the big numbers in *wei*; however, there is still an upper limit to the number of significant digits that can be kept in JavaScript Numbers–that is, as values get bigger we would lose confidence in the lower digits, and there will inevitably be errors in the final values. To deal with these limitations, the system needs to permit a certain level of errors (or ε) in exchange rates in the verification.

When determining the validity of ΔX and ΔY, the smart contract employs the Ratio Comparison method, which compares the theoretical (expected) and empirical (actual) values of the exchange rate. The theoretical rate is driven purely from the equations, whereas the empirical rate is driven from the empirical values. The two values can be represented as $\frac{R(X)+R(X+\Delta X)}{2}$ and $\frac{\Delta Y}{\Delta X}$, respectively, and they are identical if there is no precision error.

Theorem 3: Without precision errors, the expected exchange rate and the actual exchange rate are identical, i.e.,

$$\frac{R(X)+R(X+\Delta X)}{2} = \frac{\Delta Y}{\Delta X}. \tag{29}$$

Proof. By (6), $\Delta Y = \frac{\beta}{2}((X_t + \Delta X)^2 - X_t{}^2) + R_0 \Delta X$.

Therefore,

$$\frac{\Delta Y}{\Delta X} = \frac{1}{\Delta X}\left[\frac{\beta}{2}(X_t{}^2 + 2X_t\Delta X + \Delta X^2 - X_t{}^2) + R_0\Delta X\right] = \beta\left(X_t + \frac{1}{2}\Delta X\right) + R_0.$$

By (4),

$$\frac{R(X_t)+R(X_t+\Delta X)}{2} = \frac{(\beta X_t + R_0)+(\beta X_t + \beta \Delta X + R_0)}{2}$$
$$= \frac{2\beta X_t + \beta \Delta X + 2R_0}{2}$$
$$= \beta\left(X_t + \frac{1}{2}\Delta X\right) + R_0$$
$$= \frac{\Delta Y}{\Delta X}.$$

Q.E.D.

Restricting the rate of error in exchange rates using the Ratio Comparison method has an effect of having the relative error of ΔX, i.e., $\frac{\Delta X_\varepsilon - \Delta X}{\Delta X}$, bounded by ε.

Theorem 4: The relative error of ΔX is bounded by ε, i.e.,

$$\left|\frac{\Delta X_\varepsilon - \Delta X}{\Delta X}\right| < \varepsilon. \tag{30}$$

Proof. Let us define ΔX_ε as an ΔX value with precision errors, and the verifier function as

$$V(\Delta Y, \Delta X_\varepsilon) = \left|\frac{\frac{\Delta Y}{\Delta X_\varepsilon} - \frac{R(X)+R(X+\Delta X)}{2}}{\frac{R(X)+R(X+\Delta X)}{2}}\right|.$$

Since $\frac{R(X)+R(X+\Delta X)}{2} = \frac{\Delta Y}{\Delta X}$, $V(\Delta Y, \Delta X_\varepsilon)$ can be rewritten as $\left|\frac{\frac{\Delta Y}{\Delta X_\varepsilon} - \frac{\Delta Y}{\Delta X}}{\frac{\Delta Y}{\Delta X}}\right|$ and the Ratio Comparison method states that $V(\Delta Y, \Delta X_\varepsilon) = \left|\frac{\frac{\Delta Y}{\Delta X_\varepsilon} - \frac{\Delta Y}{\Delta X}}{\frac{\Delta Y}{\Delta X}}\right| < \varepsilon$. So,

$$-\varepsilon < \frac{\frac{\Delta Y}{\Delta X_\varepsilon} - \frac{\Delta Y}{\Delta X}}{\frac{\Delta Y}{\Delta X}} < \varepsilon$$

which means

---

[17] https://github.com/MikeMcl/bignumber.js/

$$-\varepsilon < \frac{\Delta X_\varepsilon - \Delta X}{\Delta X} < \varepsilon.$$

Q.E.D.

By comparing the exchange rates, the calculations are simplified drastically. Moreover, by comparing the relative error of the exchange rate to ε, the validity of inputs can be tested reliably with varying exchange rates. For example, had we compared the rates themselves, as the rate became a large number with many significant digits, its error would have increased linearly. In contrast, the fraction of error in the rate stays constant in the exchange rate.

With the OCOV model, there is an issue of transaction conflicts as the system can only process one swap (one transaction) at a time. If there are two users who send transactions at the same time, after one of the transactions is processed, the other transaction becomes invalid since both transactions' inputs have been calculated based on the same X_t value, which has been modified in between. Due to this problem, Fanco Swap utilizes both Newton's method and the Ratio Comparison method. For the swaps in Token-to-Ether direction, Fanco Swap takes the token amount as an input and calculates the corresponding Ether amount on-chain, using equation (6). For the Ether-to-Token direction, it takes both the token amount and Ether amount as inputs, as well as the X_t value, which is the token circulation at the moment the inputs have been calculated. When processing the swap, if X_t has been changed since the user sent the transaction, the token amount is recalculated using (8) and Newton's method. Otherwise, the inputs ΔX and ΔY can be verified with the Ratio Comparison method and are used as the amount of *FANCO* that the user would get and the amount of *ETH* that the user would pay, respectively.

4.3 Implementations

4.3.1 Constants

R0 = 23,629,374,000,000 *wei*

The initial exchange rate ($\frac{ETH}{FANCO}$) set for Fanco Swap. This is the exchange rate set at the creation of Fanco Swap and will not be altered in the lifetime of Fanco Swap. R_0 is obtained using equation (12), where X_s is the amount of token that was being circulated in the aFan app when the exchange was launched, and Y_s is the amount of Ether budget that was used to buy X_s token for reserve. For Fanco Swap, $X_s = 11,703,563\ FANCO$ and $Y_s = 333\ ETH$. We set $k = 10$ and $Y_k = 33,333\ ETH$ so that token's value increases tenfold once 33,333 *ETH* has been exchanged for token.

α = 830,000 *wei*

The α value, or the slope of the $R(X)$, is obtained from equation (13), with the same X_s, Y_s, k and Y_k values as those used for calculating R_0.

ε = 1 wei (= 10^{-18} ETH)

ε is the maximum relative error permitted in the exchange rate system. The fact that all the parameters and input values are in *wei* (i.e., multiplied by 10^{18} and decimals are discarded) has an effect of rounding values to 1 *wei*. We were able to reduce ε to less than 1 *wei*, and therefore directly compare values in the Ratio Comparison.

4.3.2 Functions

4.3.2.1 Ratio Comparison

Function verifyRatio() accepts two parameters, deltaX ($= \Delta X$) and deltaY ($= \Delta Y$), which are the change in $X(= -B)$ and the change in $x(= b)$, respectively. The function returns whether the deltaX and deltaY are valid at the moment (when $X = X_t$) for an Ether-to-Token swap. This is checked by comparing $\frac{deltaY}{deltaX}$ (= the actual rate) and $\frac{R(X) + R(X + deltaX)}{2}$ (= the expected rate), and deltaX and deltaY are said to be valid if the two rates are equal within the permitted error threshold. As Theorem 3 shows, the two rates are identical if there are no precision errors in PG-PERS($p = 1$). Table 2 shows the Solidity code for ratio verification.

Table 2. Ratio verification algorithm (verifyRatio).

```solidity
function getRateAtX(int256 X) view public returns(int256) {
    int256 C = SLOPE;
    int256 R = INTERCEPT;
    return (C.mul(X).add(R.mul(1 ether))).div(1 ether);
}


function verifyRatio(int256 deltaY, int256 deltaX) view internal {
    uint256 Xt = tokenCirculation;
    uint256 R1 = getRateAtX(Xt);
    uint256 R2 = getRateAtX(Xt.add(deltaX));
    uint256 expectedR = R1.add(R2).div(2);
    uint256 actualR = deltaY.mul(1 ether).div(deltaX);
    uint256 diff = expectedR < actualR ? actualR.sub(expectedR) : expectedR.sub(actualR);
    require(diff == 0, "Amount verification failed");
}
```

4.3.2.2 Token-to-Ether Swaps

By calling tokenToEther() function, the recipient gets deltaY *ETH* in exchange for deltaX *FANCO*, where deltaY is calculated with equation (6). The sender of the message needs to call Token.approve(exchange, deltaX) beforehand, or alternatively, s/he can make just one call to Token.approveAndCall(exchange, deltaX, data) that executes tokenToEther() function inside the method. In case of changes in X_t between the transaction submission and execution, the recipient will get at least the lowerBound amount of Ether. If the amount of Ether has dropped too low (below the lowerBound), the transaction will be reverted by intention. Table 3 shows the pseudocode for tokenToEther() function.

Table 3. Pseudocode of the tokenToEther() function.

```solidity
function tokenToEther(address _recipient, uint256 _dX, uint256 _lowerBound)
    public
    onlyWhileOpen
{
    checkTokenAmount(_dX);
    require(tokenCirculation >= _dX, "token amount exceeds tokenCirculation");
    /// Calculate roundedDY, the amount of ETH to give to the user.
    checkEtherAmount(roundedDY);
    /// Check that roundedDY meets the _lowerBound requirement.
    /// Check that the contract has enough ETH.
    /// Update TOKEN and ETH amounts in the contract.
    /// Transfer _dX amount of TOKEN from _recipient to this swap contract.
    /// Transfer roundedDY amount of ETH to _recipient.
}
```

4.3.2.3 Ether-to-Token Swaps

By calling eitherToToken() function, the recipient gets deltaX *FANCO* in exchange for msg.value amount of *ETH*. This function first checks whether the input Xt equals the current value of tokenCirculation (X_t), and if so, uses verifyRatio() to check the validity of the inputs deltaX and deltaY. User needs to send Ether along with the transaction as msg.value, which should be the rounded up value of deltaY. If the two values do not match up, the Newton's method kicks in, calculating a new token amount with msg.value and tokenCirculation. If the amount of token user would get is lower than

lowerBound, the transaction will be reverted by intention. The pseudocode for the function is shown in Table 4.

Table 4. Pseudocode of the etherToToken() function.

```
function etherToToken(
    address _recipient,
    uint256 _dX,
    uint256 _dY,
    uint256 _Xt,
    uint256 _lowerBound
)
    public
    payable
    onlyWhileOpen
{
    checkEtherAmount(msg.value);
    /// Define dX as the TOKEN amount that user would get,
    /// rounded to the smallest unit of TOKEN.
    checkTokenAmount(dX);
    /// If the exchange rate stayed the same
    /// (no other transactions have been processed in between),
    /// verify using verifyRatio() and use the dX and dY provided by the client.
    /// Otherwise, use the dX recalculated from calcTokenAmount().
    /// Check that dX meets the _lowerBound requirement.
    /// Update TOKEN and ETH amounts in the contract.
    /// Transfer dX amount of TOKEN to _recipient.
}
```

5. Conclusion

In this paper, we proposed a new exchange rate scheme, named polynomial exchange rate scheme (PERS), that defines exchange rates as polynomial functions of the total balances of tokens, and provided a proof that the exchange rate functions generated by the scheme are consistent, stable, and resilient. Compared with the existing deposit-based exchange rate schemes, PERS has advantages that 1) it requires no initial key token deposit and 2) it has relatively stable price change rates especially when the total balances approach extreme cases.

As a real-world application of PERS, we presented Fanco Swap where users can swap Fanco token with Ether coin and vice versa. Fanco Swap is a PG-PERS($p = 1$) system, meaning it uses PERS to determine the price and guarantees Ether payments in exchange for the token in circulation by applying the SCS principle, which limits the source of Fanco token to only Fanco Swap.

By the design constraints of Solidity–EVM framework, on which the presented swap system is implemented, we encountered several issues such as 1) precision problem and 2) computation cost problem. These issues could be well addressed by adopting the off-chain computation on-chain verification (OCOV) model and the Ratio Comparison method in addition to Newton's method for square roots. As a result, the epsilon parameter (ε) needed in the Ratio Comparison could be lowered to the level of 10^{-18} *ETH* (1 *wei*).

We hope that the proposed scheme and the presented solutions to the practical issues would be widely adopted to other token swap systems in the future.

6. Future Work

As follow-up research topics, we have several ideas listed below:

1. Provide more in-depth analysis of the relationship between the token price volatility and the parameters of PG-PERS

2. Use PERS as a model to analyze the price changes in order-book based token markets

3. Extend PERS to support direct swaps between derived tokens (e.g. between ERC20 tokens) or cross-chain cryptocurrencies

References

[1] Vitalik Buterin, "A Next-Generation Smart Contract and Decentralized Application Platform," 2013, Available: https://github.com/ethereum/wiki/wiki/White-Paper.

[2] JT Hamrick, Farhang Rouhi, Arghya Mukherjee, Amir Feder, Neil Gandal, Tyler Moore, and Marie Vasek, "The Economics of Cryptocurrency Pump and Dump Schemes," Workshop on the Economics of Information Security, 2019.

[3] Tier Nolan, "Alt Chains and Atomic Transfers", Bitcoin Forum, 2013. Available: https://bitcointalk.org/index.php?%20topic=193281.0.

[4] Mahdi H. Miraz and David C. Donald, "Atomic Cross-chain Swaps: Development, Trajectory and Potential of Non-monetary Digital Token Swap Facilities," Annals of Emerging Technologies in Computing (AETiC) Vol. 3, No. 1, 2019, Available: http://aetic.theiaer.org/archive/v3/v3n1/p5.pdf.

[5] Eyal Hertzog, Guy Benartzi, and Galia Benartzi, "Bancor Protocol: Continuous Liquidity for Cryptographic Tokens through their Smart Contracts," 2018, Available: https://storage.googleapis.com/website-bancor/2018/04/01ba8253-bancor_protocol_whitepaper_en.pdf.

[6] Meni Rosenfeld, "Formulas for Bancor System," 2016, Available: https://drive.google.com/file/d/0B3HPNP-GDn7aRkVaV3dkVl9NS2M/view.

[7] Emin G. Sirer and Phil Daian, "Bancor Is Flawed," Hacking Distributed, Available: http://hackingdistributed.com/2017/06/19/bancor-is-flawed/.

[8] Yi Zhang, Xiaohong Chen, and Daejun Park, "Formal Specification of Constant Product ($x * y = k$) Market Maker Model and Implementation," 2018, Available: https://github.com/runtimeverification/verified-smart-contracts/blob/uniswap/uniswap/x-y-k.pdf.

[9] "Solidity," Read the Docs, Available: https://solidity.readthedocs.io/en/latest/.

[10] Gavin Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," Byzantium Version, 2019, Available: https://ethereum.github.io/yellowpaper/paper.pdf.

[11] "Newton's Method for Integer Square Root," Wikipedia, Available: https://en.wikipedia.org/wiki/Integer_square_root.