PLUTON REWARD EMISSION

Plutus.it Yellow Paper - v. 1.0

ABSTRACT: *Initial design for Pluton reward emission was first set out in the white paper, however, several practical factors have influenced its realisation. Here we highlight the relevant changes that were necessary to make the reward system functional and ready for production.*

1. Introduction

Pluton Yellow Paper outlines necessary adjustments in the reward calculation and finalizes the reward system in a way suitable for production-ready implementation. To incorporate new developments in Bitcoin, Ethereum, and its markets, scaling adjustments have been made to ensure the longevity of the token.

Note that these adjustments are proportional and simply account for more volume, and no changes have been made to the fundamental structure of the reward system. In essence, this is simply to ensure that the behavior of the token matches the expectations outlined in the White Paper [1].

2. Pluton Token Infrastructure

Plutons (PLU) are defined as the internal digital currency of Plutus and are issued on the Ethereum blockchain at the Pluton contract address [2] with 18 token decimals. This also means that as a standard ERC20 token, the information necessary to incorporate Pluton in any third-party exchange or cryptocurrency service is publicly available. And because Pluton is a fully decentralized token, Plutus has no influence on how Pluton is used outside of our software system. (see Appendix-A: Source code for Pluton smart contract)

Within Plutus itself, Plutons can be used to converted into payments card and trades, with the added benefits of free conversion (0% Fee) and instant confirmations. In several Plutus updates and publications, this is usually referred to as 'priority' or 'VIP' due to an emphasis on user-side convenience. Fees when using Plutons are always 0% and will remain that way throughout the platform.

For purposes of trading, charging a balance, and other operations, Ethereum's support for near real-time confirmations (PoW: 1 Block / ~12 secs) makes practical applications feasible.

The following source code implements standard token functionality. Please note that reward emissions will be handled by a separate (funded) smart contract.

3. Pluton Ownership & Reward Emissions

Ownership of Pluton is tied to directly an Ethereum address and its private key. Smart contracts, running on the production Ethereum-network (network-id 1), act effectively as a trustless notary that tracks token ownership, trades and transactions. As such, it benefits from the distributed computing and block-finding incentive model, which ensures the integrity and security of the ledger itself and that of its tokens.

The updated reward system has a scale from 30 to 150 BTC. This is proportional to the scale of 5-25 used in the White Paper, albeit adjusted for higher volumes. In the previous calculator (assuming 5-25 BTC), each step resulted in a 0.1 increase in volume and a 0.01% decrease in rewards. Whereas, in the new calculations (now assuming 30-150 BTC),

each step results in a 0.1 increase in volume and a 0.0165 % decrease in rewards

Similar to the whitepaper, a transaction in the first 20% volume gets 3% reward. In the new scale this translates to ≤ 30 BTC. To maintain a healthy 120-150 years longevity, the required BTC:PLU ratio is $\sim 1:250$.

4. Plutons Reward System

Plutus app users are rewarded Plutons for every cryptocurrency (private wallet balance) to fiat (Plutus contactless payments balance) conversion made using the Plutus app or card. However, as a logical consequence of reward emissions, no reward occurs when making a Pluton to fiat conversion. Pluton can be used to charge a fiat balance immediately, without any delays or conversion fees.

The reward system is a smart contract on Ethereum network and triggered by the PlutusDEX-contract. Only PlutusDEX-contracts can trigger the reward system to distribute the intended amount of Pluton to app and card user(s) automatically.

Out of 20,000,000 total supply, 850,000 Plutons were distributed during the token sale. Trading on the DEX network, under the exchange rate, E_P, will fluctuate according to market conditions. Its value is determined by the amount of Pluton, P_t, required to trade for one Bitcoin, or

$$E_P = \frac{P_t}{1\ BTC} \tag{1}$$

The DEX user (trader) will be able to buy Plutons as well as Bitcoins and Ether. Pluton rewards are limited to in app uses only, which can be used to transfer to another user or redeem using contactless payments balance for in-store purchases. Pluton is implemented as a 'transferable fungibles' i.e. sub-currency on Ethereum [3] .

Since rebate rewards are only dispersed when converting Bitcoin to fiat money we must utilize the variable exchange rate of, E_{BTC}, Bitcoin to the British Pounds, expressed as

$$E_{BTC} = \frac{1\ BTC}{T_i\ GBP} \tag{2}$$

Let T_i symbolize a single transaction by a single user worth no more than a maximum value of £30. Multiplying T_i by the variable Bitcoin exchange rate yields the Bitcoin value, B_i, of the given transaction,

$$B_i = E_{BTC} * T_i \tag{3}$$

where $T_i \leq £30$.

The reward rate, R, changes every 24 hours, at noon GMT, according to the previous day's aggregate Bitcoin transaction volume on the PlutusDEX, V^{Δ}, expressed as

$$V^{\Delta} = \sum_{i=1}^{n} B_i^{\Delta} \tag{4}$$

where the delta (Δ) in the superscript refers to data from the previous day. As seen above, the previous day's aggregate Bitcoin transaction volume is obtained by summating each instance of equation (3) on the previous day.

The reward rate, R, is a dependent variable to V^{Δ} by adapting and modifying the model of a step function to a non-Boolean use. The indicator function of R is defined as the interval expression, $I_{V^{\Delta}}$, with the subscript, V^{Δ}, of equation (4), which acts as the interval parameter function of the next day's reward rate, defined by the expression

$$R := I_{V^{\Delta}}(R) \tag{5}$$

The interval parameter function is defined by the variable values of R, which are dependent on the corresponding range of the previous day's Bitcoin transaction volume, V^{Δ}.

$$I_{V^{\Delta}}(R) := \{ R_i, \quad m_i < V^{\Delta} \leq M_i \} \qquad (6)$$

The reward rate, R_i, in the above expression represents the general structure of the parameters that describe every possible value. Each daily reward rate, R_i, must differentiate from the previous day's rate, R^{Δ}, by ±0.0165%.

Where each step is denoted by:

$$m_i - M_i = 0.1\ BTC \qquad (7)$$

The maximum reward rate is set at 3% when the daily volume is 30 BTC or less and 1% when the daily volume is 150 BTC or more. Thus, each daily increase in volume of 0.1 BTC will reduce the reward rate by 0.0165%, and vice-versa.

Now that the reward rate, R, has been found above, we can find the amount of Pluton, P_i for one user's single transaction by multiplying the results of equation (1), equation (3), and the current exchange rate, R^{Δ}, to find

$$P_i = E_p * B_i * R^{\Delta} \qquad (8)$$

The reward rate has the delta in its superscript to indicate that it was calculated using the previous day's Bitcoin transaction volume as expressed in equation (5).

Finally, we can calculate the total Pluton dispersed to users for an entire day, P_{day}, by summation of the results obtained from equation (3) and equation (8) and simplifying to obtain the emission equation:

$$P_{day} = R * \sum_{i=1}^{n} (P_i * B_i)$$

Program steps for reward emission:

1. Aggregate transaction volume is sampled every 10 minutes.
2. Aggregate transaction volume is reset at 12.00 GMT. (Every 24 hours at noon.)
3. When a transaction (buying of fiat) is completed on the system, the Pluton reward rate is calculated in real time based on two inputs:
   a. the current aggregate transaction volume (volume calculated in last 10 minutes)
   b. the current sampled market price of Pluton
4. The real-time market price is based on several exchange rates (ideally weighted, similar to BTC and ETH).
5. After the reward has been calculated, the Plutons amount is credited to the user's Pluton balance directly.
6. When the user chooses to withdraw, the Plutons are transferred to the user's preferred wallet.

4. Pluton Reward Emission Calculator

For ease of use, the Pluton reward calculator has been released. This web application lets users estimate how many Pluton will be emitted over time. A live version of the newly updated Pluton reward calculator is made available at the following link:

https://plu-calc-xvavrkviyp.now.sh/
(under development)

Enter different values on the right to explore how Pluton will behave when making a deposit. If the link is currently down, please contact support@plutus.it or try again later.

Although, the calculator is initialised using current market price for both BTC and PLU, these fields accept speculative inputs from users

Sample real life scenarios calculated using the calculator:

(a) User makes a deposit of 0.01 BTC, which at a bitcoin price of $10,000 is $100. This transaction incurs a fee of $3.

(b) Assuming PLU is $20 and the total volume on the PlutusDEX is lower or equal to 30 BTC (which sets the reward rate to 3% per

deposit), the user also receives a reward of 0.15 PLU.

5. Performance, scalability & reliability

Unlike MVPs, production systems need to be robust and reliable while at the same time deliver performance under increasing loads. PlutusDEX uses high-performance BlockCypher explorer nodes that not only support the latest features of Bitcoin and Ethereum networks but also facilitate simultaneous or parallel processing of transactions in both these networks. As a result, the PlutusDEX is able to process 12 transactions per second, which is faster than Bitcoin and matching the speed of Ethereum. Since BlockCypher maintains several blockchain nodes in a high-availability architecture, they guarantee PlutusDEX can scale with increasing number of users and transactions with little or no changes to the system configuration. With regards to scalability changes in Bitcoin protocol, PlutusDEX will support Segwit transactions [4] .

The current transaction fee for Bitcoin starts at $4-6 and even reaches $20, especially when there is a huge backlog transactions waiting to be confirmed. In such cases, small valued transactions on our platform will become prohibitive until wallets add native mainnet Lightning network [5] support. Similarly, Plutus will explore the use of Raiden network [6] to reduce gas costs for Ethereum transactions.

6. Conclusion

Here the changes to the initial Pluton reward emission has been highlighted and discussed, this included changes to scaling factor, design changes, exploiting of 3rd-party services and networks. In the next phase of software development, these changes will be realised and the community will be informed.

References:

[1] Plutus White Paper (2016). Available at: https://plutus.it/public/white-paper.pdf.

[2] Pluton Smart Contract. Available at: 0xD8912C10681D8B21Fd3742244f44658dBA12264E

[3] Ethereum Foundation (2015) ERC-20 Token Standard. Available at: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md (Accessed: 28 February 2018).

[4] SegWit (no date). Available at: https://en.wikipedia.org/wiki/SegWit (Accessed: 28 February 2018).

[5] Lightning Network (no date). Available at: https://lightning.network/ (Accessed: 1 March 2018).

[6] Raiden Network (no date). Available at: https://raiden.network/ (Accessed: 1 March 2018).

GLOSSARY

Cryptocurrency: Decentralized token, exclusively refers to Bitcoin, Ethereum, and Pluton at the time of writing. May change in the future.

ERC20: Ethereum Request for Comments. This is an official protocol for proposing improvements to the Ethereum network. '20' is the unique proposal ID number. ERC20 defines a set of rules for standard blockchain tokens, making them easier to interact with and ensuring portability.

DEX user or Trader: A buyer of cryptocurrencies on the PlutusDEX.

MVP: minimum viable product or proof-of-concept application

PLU: Pluton, the loyalty rewards utility token of Plutus.

User: A user of the Plutus app and/or Plutus Debit Card. Typically owns cryptocurrencies already.

APPENDIX - A Source Code for Pluton Smart Contract

```
/*
The Pluton Contract implements the standard token functionality
(https://github.com/ethereum/EIPs/issues/20) as well as the following OPTIONAL extras
intended for use by humans.

Pluton contract extends HumanStandardToken, https://github.com/consensys/tokens

.*/
contract Token {

    /// @return total amount of tokens
    function totalSupply() constant returns (uint256 supply) {}

    /// @param _owner The address from which the balance will be retrieved
    /// @return The balance
    function balanceOf(address _owner) constant returns (uint256 balance) {}

    /// @notice send `_value` token to `_to` from `msg.sender`
    /// @param _to The address of the recipient
    /// @param _value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transfer(address _to, uint256 _value) returns (bool success) {}

    /// @notice send `_value` token to `_to` from `_from` on the condition it is approved
by `_from`
    /// @param _from The address of the sender
    /// @param _to The address of the recipient
    /// @param _value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transferFrom(address _from, address _to, uint256 _value) returns (bool
success) {}
```

```
    /// @notice `msg.sender` approves `_addr` to spend `_value` tokens
    /// @param _spender The address of the account able to transfer the tokens
    /// @param _value The amount of wei to be approved for transfer
    /// @return Whether the approval was successful or not
    function approve(address _spender, uint256 _value) returns (bool success) {}

    /// @param _owner The address of the account owning tokens
    /// @param _spender The address of the account able to transfer the tokens
    /// @return Amount of remaining tokens allowed to spent
    function allowance(address _owner, address _spender) constant returns (uint256
remaining) {}

    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}


contract StandardToken is Token {

    function transfer(address _to, uint256 _value) returns (bool success) {
        //Default assumes totalSupply can't be over max (2^256 - 1).
        //If your token leaves out totalSupply and can issue more tokens as time goes on,
you need to check if it doesn't wrap.
        //Replace the if with this one instead.
        //if (balances[msg.sender] >= _value && balances[_to] + _value > balances[_to]) {
        if (balances[msg.sender] >= _value && _value > 0) {
            balances[msg.sender] -= _value;
            balances[_to] += _value;
            Transfer(msg.sender, _to, _value);
            return true;
        } else { return false; }
    }

    function transferFrom(address _from, address _to, uint256 _value) returns (bool
success) {
        // same as above. Replace this line with the following if you want to
        // protect against wrapping uints.
        if (balances[_from] >= _value && allowed[_from][msg.sender]
>= _value && _value > 0) {
            balances[_to] += _value;
            balances[_from] -= _value;
            allowed[_from][msg.sender] -= _value;
            Transfer(_from, _to, _value);
            return true;
        } else { return false; }
    }

    function balanceOf(address _owner) constant returns (uint256 balance) {
        return balances[_owner];
    }

    function approve(address _spender, uint256 _value) returns (bool success) {
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
        return true;
    }

    function allowance(address _owner, address _spender) constant returns (uint256
remaining) {
      return allowed[_owner][_spender];
    }
```

```
    mapping (address => uint256) balances;
    mapping (address => mapping (address => uint256)) allowed;
    uint256 public totalSupply;
}


contract HumanStandardToken is StandardToken {

    function () {
        //if ether is sent to this address, send it back.
        throw;
    }

    /* Public variables of the token */

    /*
    NOTE:
    The following variables are OPTIONAL vanities. One does not have to include them.
    They allow one to customise the token contract & in no way influences the core
functionality.
    Some wallets/interfaces might not even bother to look at this information.
    */
    string public name;                   //fancy name: eg Simon Bucks
    uint8 public decimals;                //How many decimals to show. ie. There could 1000
base units with 3 decimals. Meaning 0.980 SBX = 980 base units. It's like comparing 1 wei
to 1 ether.
    string public symbol;                 //An identifier: eg SBX
    string public version = 'H0.1';       //human 0.1 standard. Just an arbitrary
versioning scheme.

    function HumanStandardToken(
        uint256 _initialAmount,
        string _tokenName,
        uint8 _decimalUnits,
        string _tokenSymbol
        ) {
        balances[msg.sender] = _initialAmount;               // Give the creator all
initial tokens
        totalSupply = _initialAmount;                        // Update total supply
        name = _tokenName;                                   // Set the name for display
purposes
        decimals = _decimalUnits;                            // Amount of decimals for
display purposes
        symbol = _tokenSymbol;                               // Set the symbol for display
purposes
    }

    /* Approves and then calls the receiving contract */
    function approveAndCall(address _spender, uint256 _value, bytes _extraData) returns
(bool success) {
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);

        //call the receiveApproval function on the contract you want to be notified. This
crafts the function signature manually so one doesn't have to include a contract in here
just for this.
        //receiveApproval(address _from, uint256 _value, address _tokenContract, bytes
_extraData)
        //it is assumed that when does this that the call *should* succeed, otherwise one
would use vanilla approve instead.
```

```
if(!_spender.call(bytes4(bytes32(sha3("receiveApproval(address,uint256,address,bytes)"))),
msg.sender, _value, this, _extraData)) { throw; }
        return true;
    }
}

// Creates 20,000,000.000000000000000000 Pluton (PLU) Tokens
contract Pluton is HumanStandardToken(20000000000000000000000000, "Pluton", 18, "PLU") {}.
```

Also viewable on:

https://etherscan.io/address/0xD8912C10681D8B21Fd3742244f44658dBA12264E#code