Cardstack Token Mechanism

Chris Tse, Hassan Abdel-Rahman, Justin Thong

November 28, 2017

Abstract

This paper clarifies the token mechanism for the Cardstack ecosystem. In the introduction, common keywords are defined and notational conventions explained. The main section focuses on the blockchain and describes the different time instances, oracle functions, and contract types. The section on the token mechanism introduces the concept of 4R and examines each R in detail in the form of pseudocode – showing how all the contracts and oracles communicate once an event is triggered.

1 Introduction

1.1 Main Definitions

- **Cardstack Application**: A *Cardstack Application* is a subscription of a service to the open-source Cardstack Software Layer. This layer makes it possible for applications to be built in connection with hosting services that are available on the cloud, such that users of the Cardstack Application are able to maintain sovereignty over their data and software. The Cardstack Application is then represented and governed by a Cardstack Application Smart Contract.

- **Cardstack Users and Maintainers**: A *Cardstack User* is the owner of the Cardstack Application and contributes monetary wealth towards the payment of services to the Cardstack ecosystem in the form of Cardstack Tokens (CST). A *Cardstack Maintainer* is a developer who has a contractual agreement with the Cardstack User, is typically deemed to have contributed some intellectual work in terms of the development of a Cardstack Application, and is paid fixed fees. There may be more than one Cardstack Maintainer, hence forming the primary application team. A Cardstack User is not eligible for a reward, whereas Cardstack Maintainers are eligible for rewards equivalent to the ones received by open-source contributors.

- **Cardstack Token (CST)**: The *Cardstack Token* is an ERC20-compliant token that carries inherent value and is used to acquire Software and Services Coupons (SSC).

- **Software and Services Coupon (SSC)**: The *Software and Services Coupon* is a service coupon that is pegged to the value of the USD and can be acquired using Cardstack Tokens (CST), depending on the CST-to-SSC exchange value. SSC are used to redeem services for a Cardstack Application, such as hosting services and services for the payment of network fees.

- **Open-Source Project/Contributors**: An open-source project is a publicly accessible repository of written and working code that has its own Github and npm address[1]. Each open-source project has its own project dependency tree structure and a single public Ethereum address associated with it. An open-source project must be connected to a particular Cardstack Application (which is an open-source project itself) through a dependency structure, in order to be eligible to receive rewards.

- **Reward**: A reward is a grant of Cardstack Tokens (CST) to open-source projects associated with a particular Cardstack Application. It is given based on the advice of the attribution

[1] npm provides a facility to obtain a project dependency tree. npm may be substituted by another software in the future

oracle, which accounts for signals that deem an open-source project worthy of a reward. The minting of new CST in the reward pool as well as unprocessed CST stemming from expired SSC may also be used to give rewards to these projects.

- **Reward Pool:** A reward pool is a store of CST that originates from the conversion of CST to SSC. The size of the reward pool serves as an indicator of economic activity [2]; therefore, it is used in the process of minting new CST. The reward pool, once collected, is used to reward participating open-source projects. It also acts as vehicle for payment of material cost to service providers (hosting services and Cardstack Maintainers) as well as network fees.
- **Proof-of-Burn:** Proof-of-Burn – also referred to as burn signal – is the evidence of the redemption of Cardstack Application services, through the expenditure of SSC by a Cardstack Application, over a period of time. A big amount of SSC burned means that an application is economically active at the time of usage; ideally, a large number of SSC burned by the network as a whole should be reflected in the size of the reward pool.
- **Smart Contracts:** Smart contracts are programmatic functions inside the blockchain that record and execute the logic of a contractual agreement. They act, to an extent, as an immutable memory, storing information that helps them carry out their function. A contract's logic can be exercised in two ways: An external user activates it or an event from another contract triggers it. These operations, along with any changes made to a contract's state, require the payment of a small fee (gas).
- **Oracle:** The oracle function is a function that exists off-chain and reports the value(s) of an event at a certain point of time. The oracle function interacts with the blockchain, ensuring that more complex operations can be executed; otherwise, the operations in the blockchain would be limited. Oracles are commonly utilised to activate smart contracts through spending from wallets to which they have access.
- **Token Generation Event (TGE):** A Token Generation Event is an occasion, taking place over a period of time, during which the Cardstack Foundation makes available a percentage of the total amount of CST that have been minted from the start. This happens via distribution of new tokens from a token-issuing smart contract (CST Smart Contract –see section 2.3.1).

1.2 Notation

In this section, we explain the notation for smart contracts and oracle functions. We describe a contract through a function (denoted by C), which takes three different inputs. The first input is I_1: The set of parameters for the contract at the onset of its creation. The second input is I_2: The set of contracts providing values that can be read (CALL). The third input is I_3: The set of oracle functions that, once executed, report some value(s) to the smart contract. The outputs of a contract function denote the set of values that it stores or that are available once the inputs have been processed, Σ. The outputs may be thought of as parameters – the contract's state – that can be changed at any point of time. The format of a contract function is shown below:

$$C_{letter}((I_1), (I_2), (I_3)) = (\Sigma) \tag{1}$$

The oracle functions (denoted by O) assume a similar form, but without making a distinction between the types of input. The oracle function takes a set of inputs, I, and returns a set of outputs, Σ'; intuitively, this is the format of a regular function, as below. The outputs of an oracle function are different from a smart contract, in the sense that they can only be returned, but they cannot be changed like parameters can, as they do not possess a notion of state.

$$O_{letter}(I) = (\Sigma') \tag{2}$$

If there is no set of inputs, the round brackets are simply left empty. Occasionally, we exploit the use of dots $(\dots)$ to suggest that the outputs are the inputs or the inputs are implicitly available. We use typical mathematical notation to refer to a bold letter as a vector and a non-bold letter as a scalar. Also, a subscript on top of a bolded letter refers to a single component of a vector, e.g. a^i in $\mathbf{a}$. In our notational use, we attempt to assign a letter or symbol that relates to the word being referred to, e.g. O for oracle, B_E for balance of ether.

[2] although this may possibly be lagged

2 Blockchain

2.1 Time

In our token mechanism, there are different timings, overlapping with each other, that dictate when an event occurs. This section describes these various timings and how they are related.

- **Block Time, $\mathbf{t_B}$:** The *block time* is denoted by the block height, which is the number of blocks between the genesis block and the current block.
- **Reward Time, $\mathbf{t_R}$:** The *reward time* is the instance in which the reward pool is locked (and the rewards are being "prepared" to be issued) after the distribution of rewards from the previous locked reward pool is completed. Following the lock of the reward pool, a secondary reward pool is opened, which marks the beginning the next reward pool. The reward time creates the time interval (or period) in between which the previous reward pool is locked and the current reward pool is locked, $(t_R - 1, t_R)$, otherwise known as the time interval in which the previous reward pool is locked for or the time interval in which the current reward pool is unlocked. The locking period defines a period of time in which an attribution oracle needs to run and the distribution of rewards to all addresses needs to take place. A reward time interval can span more than one block time.
- **Metering Time, $\mathbf{t_\lambda}$:** The *metering time* is a "heartbeat" (triggered by a heartbeat function – see section 2.2.1) that indicates a time instance in which the metering oracle reports a value. When a reward pool is locked, the last available metering time is used to compute the proportional attribution – see section 3.2.6. The metering time is more frequent than the reward time, i.e. there is more than one "heartbeat" within a reward pool lock interval.
- **Fees Time, $\mathbf{t_f}$:** The *fees time* refers to a time on the regular calendar. It determines the due date of the payment from the Cardstack ecosystem (all applications) to service providers. The fees time can be the conventional billing period expected by service providers, e.g. monthly, quarterly, or annually. The interval between fees times is the longest, compared with all the other timings.

2.2 Oracle Functions

Oracle functions enrich the space of a decentralised application. In our context, we use several types of oracle functions to carry out the actions of our token mechanism:

- Heartbeat Function, f_λ
- Metering Oracle, O_m
- Fees Oracle, O_f
- Exchange Oracle, O_e
- Attribution Oracle, O_a

2.2.1 Heartbeat Function, $\mathbf{f_\lambda}$

A *heartbeat function* is a function that keeps track of the timing of events that require a "heartbeat". It evokes a smart contract that, correspondingly, evokes an oracle. Typically, we refer to the heartbeat function as a trigger for the metering oracle. However, it can be used to trigger a variety of other events too, both on-chain and off-chain. Since a sequence of actions on the blockchain can only be triggered by an external user, the heartbeat function spends ether from a private address with access to a wallet that has ether tokens; therefore, it can pay for gas to execute transactions (altering the state of contracts).

2.2.2 Metering Oracle, $\mathbf{O_m}$

The *metering oracle* is a function that reports how much Cardstack services cost in SSC, $\mathbf{N_{ssc}}$, by aggregating the hosting fees, $\mathbf{F}$, and a services multiplier, κ, associated with all Cardstack applications. At every t_λ, the oracle is evoked to fetch the additional hosting fees, $\mathbf{F}$, meaning the amount of hosting that has been used since $t_\lambda - 1$. The services multiplier, which is described in the terms $\mathbf{T}$, is the percentage multiplied by the hosting fees, $\mathbf{F}$. Notice that no conversion of the cost of services is necessary, because SSC are pegged to the USD, which is the unit used to report hosting fees.

$$O_m(\mathbf{a}, \kappa) = (\mathbf{F}, \mathbf{N_{SSC}}) \tag{3}$$

- Addresses of Cardstack applications, $\mathbf{a}$
- Services multiplier, κ
- Hosting fees since the last metering time (USD/SSC), $\mathbf{F}$
- SSC that should be burnt ($(1 + \kappa) \times F^i$), $\mathbf{N_{SSC}}$

2.2.3 Fees Oracle, $\mathbf{O_f}$

The *fees oracle* is a function which computes and reports the quantity of unpaid fees that have to be paid by the Cardstack ecosystem (as a whole); this includes the payment of material cost to service providers (hosting providers, Cardstack Maintainers) and network fees. The fees oracle is different from the metering oracle in two ways: 1) The unpaid fees of the fees oracle, in interval (t_f,$t_f + 1$), should be obtained by summing up the cost of services reported by the metering oracle at each t_m within the interval, then summing up those fees for ALL Cardstack applications in the network (assuming that all payments in the past were up-to-date). 2) The output of the metering oracle is computed by a metering prediction formula, whereas the output of the fees oracle is the actual amount of fees that has to be paid from the perspective of the providers – it is the actual bill for material fees. We expect the fees oracle to be consistent with the metering oracle up to a level of uncertainty. The main purpose of the fees oracle is to make bulk payments to service providers by the group of Cardstack applications possible. Therefore, a benefit is created for the ecosystem, as costs are reduced, while the primary application team (Cardstack Maintainers) and open-source projects are enabled to reclaim the excess payment through rewards in the form of CST.

$$O_f(\mathbf{a}) = (UF) \tag{4}$$

- Addresses of Cardstack applications, $\mathbf{a}$
- Unpaid fees (USD/SSC), UF

2.2.4 Exchange Oracle, $\mathbf{O_e}$

The *exchange oracle* is an oracle that determines the market exchange rates of CST to ether and USD, respectively. The market rates are investigated by surveying exchanges that carry CST. By calling the exchange oracle, the Cardstack Token Smart Contract and the CST->SSC Smart Contract determine the contract rates at which CST are acquired and distributed.

$$O_e(EX) = (P_{Es}, P_{Eb}, P_{USs}, P_{USb}) \tag{5}$$

- Feed from relevant market exchanges that carry CST, EX
- The market rate of acquiring 1 CST, in ether (in units of wei), P_{Es}
- The market rate of distributing 1 CST, in ether (in units of wei), P_{Eb}
- The market rate of acquiring 1 CST, in USD, P_{USs}
- The market rate of distributing 1 CST, in USD, P_{USb}

2.2.5 Attribution Oracle, $\mathbf{O_a}$

The attribution oracle is the most complex oracle of all. It consists of various factors: computations that account for the history of the application's burn signals, $\mathbf{N_{bSSC}}$; applications' opinions of other applications, $\mathbf{U}$; a priori model of a fair distribution, based on graph analysis, obtained from the npm and Github addresses, $\bar{\mathbf{a}}$; and a diversity reward, which is a global reward to reduce the inequality of wealth within the network. The purpose of this oracle is to compute the portion of the reward pool that is allocated to each address in network, $\mathbf{w}$. Underneath the computation, there are several subsidiary actions. For example: In order to account for the portion of the reward pool that should be minted, the attribution oracle reads the signals concerning the expired SSC, $\mathbf{B_{eSSC}}$, as recorded in the SSC Smart Contract; it consults the exchange oracle on the cumulative value of those signals in CST; and it communicates the value of expired SSC in CST to the Reward Smart Contract – see section 3.2.6. Occasionally, the attribution oracle may filter addresses; either with the aim to remove addresses that are spam, i.e. addresses that were artificially created, or when the cost (in gas) of sending the rewards to these addresses exceeds the rewards themselves. For more information about the attribution oracle, please refer to the Reward Paper

$$O_a(\mathbf{a}, \bar{\mathbf{a}}, \mathbf{U}, \tau, \mathbf{N_{bSSC}}, \mathbf{B_{eSSC}}) = (\mathbf{ra}, \mathbf{w}) \tag{6}$$

- Cardstack Application addresses, $\mathbf{a}$
- Associated Github profiles and npm addresses, $\bar{\mathbf{a}}$
- Weights of a Cardstack Application's first-level dependencies, recommended by the Cardstack Application itself (also known as the data set of user attributions), $\mathbf{U}$ [3]
- Percentage of the rewards taken by the primary application team (vector), τ
- History of burn signals for all applications, $\mathbf{N_{bSSC}}$
- Burn signals of expired SSC (in batches), $\mathbf{B_{eSSC}}$
- Any rewarded addresses (including open-source projects and primary application teams, after the removal of filtered addresses), $\mathbf{ra}$
- The flat version of weights, such that $w^i \times R$ denotes the reward obtained by address ra^i, $\mathbf{w}$

2.3 Contract Types

There are 8 contract types. All of these contracts interact with each other; with the help of oracle functions, they carry out the token mechanism of the 4Rs: Register, Retain, Redeem, Reward – see section 3.1.

- Cardstack Token Smart Contract, C_c
- Software and Services Coupon Smart Contract, C_s
- Cardstack Application Genesis Smart Contract, C_g
- Cardstack Application Smart Contract, C_a
- Cardstack CST->SSC Exchange Smart Contract, C_e
- Cardstack Reward Smart Contract, C_r
- Cardstack CST Custodial Smart Contract, C_{cus}
- Cardstack Team Smart Contract, C_T
- SSC Top-Off Smart Contract, C_{top}

[3] Note: $\mathbf{U}$ is not particularly a vector; it is a vector of vectors, $\mathbf{u^i}$, where all components have differing lengths.

2.3.1 Cardstack Token Smart Contract

The Cardstack Token Smart Contract governs the issuance, acquisition, and distribution of Cardstack Tokens (CST) and is owned by the Cardstack Foundation. At the start of the creation of the ecosystem, 10 billion CST are minted. These 10 billion CST are allocated for different purposes at different points in time, as specified in the whitepaper; therefore, not all CST are available immediately after the minting process. This paper covers the allocation of tokens through the Token Generation Event only. A Cardstack User acquires CST from the Cardstack Token Smart Contract via the Token Generation Event.

Parameters on issue, I_1:

- Total amount of CST available in the network, N_{CST}
- Cap on the amount of CST that can be acquired from the CST Smart Contract, M_{CST}
- Contract rate of distributing 1 CST, in ether (in units of wei), CP_{Eb}
- Contract rate of acquiring 1 CST, in ether (in units of wei), CP_{Es}

Connected Oracles, I_3:

- Market exchange oracle, O_e

The parameters can be changed at any point in time. Particularly, contract rates may change based on outputs of the market exchange oracle. Minted CST, based on economic activity at the end of each reward cycle – which is controlled by the Cardstack Reward Smart Contract – will affect the total amount of CST available, i.e. there may be more than 10 billion CST.

$$C_c((N_{CST}, M_{CST}, CP_{Eb}, CP_{Es}), (), (O_e)) = (\dots) \tag{7}$$

2.3.2 Software and Services Coupon Smart Contract

The Software and Services Coupon Smart Contract governs the issuance, acquisition, and distribution of Software and Services Coupon (SSC). Software and Services Coupon are non-transferable coupons that are pegged to the USD. A Cardstack User can use SSC, via a Cardstack Application and the terms that govern its usage, to redeem software and services provided by the hosting service in the agreement. The SSC Smart Contract has its own pool of ether, collected from the Cardstack Foundation wallet, which is utilised by the Cardstack CST Custodial Smart Contract for the acquisition of CST. The same pool of ether may be used to control inflationary pressures regarding the exchange rate of CST to USD (or CST to SSC). For example, ethers will be used to acquire CST from market vendors to reduce the supply of CST. After the creation of the smart contract, more ethers can be added to its ether pool.

Parameters on Issue, I_1:

- Initial balance of ether at the SSC Smart Contract's address, B_E
- Time period of expiry (in block height length), T_e

Connected Contracts, I_2:

- Cardstack Application Smart Contract, C_a

Outputs:

- Active balance of ether assigned to the SSC Smart Contract's address, B_E
- Burn signals of expired SSC (in batches), $\mathbf{B_{eSSC}}$

$$C_s((B_E, T_e), (), ()) = (B_E, \mathbf{B_{eSSC}}) \tag{8}$$

2.3.3 Cardstack Application Genesis Smart Contract

The Cardstack Application Genesis Smart Contract is used as a means to produce instances of the Cardstack Application Smart Contract. A Cardstack Maintainer can invoke a function in the Cardstack Application Genesis Smart Contract, containing various parameters that govern the contract between a Cardstack Application and the Cardstack ecosystem – its general terms, T_g, also known as the franchisee agreement – to create a new instance of a Cardstack Application Smart Contract. An example of the parameters specified in these general terms would be the minimum amount of SSC, B_{minSSC}, and the frequency of SSC redemptions, f_{rSSC}, required to maintain the Cardstack Application. Additionally, Cardstack Maintainers can also stipulate the non-general terms between them and the Cardstack Users via input parameters of the same function (for notational convenience, we define these input parameters in the Cardstack Application Smart Contract). If not specified, these parameters take the default value as defined in the placeholder structure. The Cardstack Application Genesis Smart Contract acts as the registry of all Cardstack Application addresses, $\mathbf{a}$. To instantiate an application contract, the Cardstack Maintainer pays a fee in CST[4]. All the fees are collected in a balance recorded by the Cardstack Application Genesis Smart Contract, B_{tpCST}, also known as the toll pool; and the foundation will source from the pool, as described in the white paper.

Parameters on issue:

- General terms of the Cardstack Application, T_g
  - Minimum amount of SSC required, B_{minSSC}
  - Frequency of SSC redemptions based on the time period, f_{rSSC}
  - etc.

Outputs:

- Cardstack Application addresses, $\mathbf{a}$
- Balance of the toll pool, B_{tpCST}

$$C_g((T_g), (), ()) = (\mathbf{a}) \rightarrow C_a \tag{9}$$

2.3.4 Cardstack Application (Debit Card) Smart Contract

The Cardstack Application Smart Contract is used to collect Software and Services Coupons (SSC) from Cardstack Users as prepayment for software and services, as outlined by the smart contract. A Cardstack Application Smart Contract is instantiated from the Cardstack Application Genesis Smart Contract, where the Cardstack Application Maintainer can stipulate the non-general terms of the agreement, as described in the previous section. A Cardstack User can validate the application by adding an amount of SSC to the Cardstack Application, taking into account the minimum SSC balance required, in the T_g. The balance of SSC in the contract can be updated at any point of time by the Cardstack User or the SSC Top-Off Smart Contract. The Cardstack Application Smart Contract burns an amount of SSC, as consulted by the metering oracle function, and records the history of burns to redeem a service [5]. The history of SSC burns, N^i_{bSSC}, stored in this contract is used by the attribution oracle to help determine the portion of the reward pool that is allocated to each open-source contributor. The Cardstack Application can be used to instantiate the Cardstack Team Smart Contract – see section 2.3.8. Any incoming rewards to the Cardstack Application Smart Contract can be divided, as described in the terms $\mathbf{T}$, and distributed to Cardstack Maintainers' addresses via the Cardstack Team Smart Contract. Otherwise, the sole Cardstack Maintainer will obtain all the rewards accumulated by the Cardstack Application. The weight of the Cardstack Application's first-level dependencies, $\mathbf{u^i}$, is the application's evaluation of how much they contribute, relatively to each other. The Cardstack Application thus makes a recommendation about how a reward, which is passed to all the first-level dependencies, should be

[4] This fee is considered part of the network fees.

[5] Since SSC are created and deposited in batches, they expire in batches too. The batch of SSC that exists in the Cardstack Application Smart Contract's balance expires in the SSC Smart Contract upon being burned.

divided. τ^i is the proportion of rewards taken by the primary application team before the rewards are passed down to the first-level dependencies.

Parameters on issue, I_1:

- Terms of the agreement, as described in previous sections (including T_g), T^i

Connected Contracts, I_2:

- Cardstack Application Genesis Smart Contract, C_g

Connected Oracles, I_3:

- Metering oracle, O_m

Outputs:

- Maintainer's address, a_m^i
- Associated Github Profile and npm address (identifier of the online profiles that contain Ethereum addresses), $\bar{a}^i$
- Active balance of SSC, B_{SSC}^i
- History of SSC burns over redemptions of service (also considered as the cost of SSC reported by the metering oracle), N_{bSSC}^i
- Proportion of rewards taken by projects, τ^i
- Weights of the Cardstack Application's first-level dependencies, $\mathbf{u^i}$

$$C_a((T^i),(C_g),(O_m)) = (a_m^i, B_{SSC}^i, N_{bSSC}^i, \tau^i, \mathbf{u^i}) \xrightarrow{\text{(optional)}} C_T \tag{10}$$

2.3.5 Cardstack CST->SSC Exchange Smart Contract

The Cardstack CST->SSC Exchange Smart Contract allows users to exchange CST for SSC, based on the current market rate of CST. The contract rates established are informed by the exchange oracle function, O_e, i.e. the contract rates are the (reported) market rates. For example, the rate of distributing 1 CST for SSC, CP_{SSCb}, is the product of the rate of distributing 1 CST for ether, P_{Eb}, and the rate of selling ether for 1 USD. The CST collected from the contract will be instantly added to the CST reward pool maintained by the Cardstack Reward Smart Contract. The expiry time begins when the SSC are created, i.e. when the SSC are assigned to a user and recorded in the SSC Smart Contract.

Outputs:

- Contract rate of distributing 1 CST, in SSC, CP_{SSCb}

$$C_e((),(),(O_e)) = (CP_{SSCb}) \tag{11}$$

2.3.6 Cardstack Reward Smart Contract

The Cardstack Reward Smart Contract is used to disseminate CST rewards based on attribution signals, as interpreted by the attribution oracle. Throughout time, the reward pool, R_t, governed by this contract collects the CST from the exchange oracle. At each t_R, the reward pool is locked at balance R_{t_R} and prepared for distribution, while another reward pool, $R_{(t>t_R)}$, is created to collect CST. Following the reward pool lock, the computation of the attribution oracle and the distribution of rewards to addresses commence. The current economic activity (of the unlocked reward pool), as charged by the gas in the mining function of the CST Smart Contract, will fund the computations and distribution in the Cardstack Reward Smart Contract (the rewards that are currently locked); this also means that the time until the lock of the next reward pool is dependent on the economic activity. This contract consults the attribution oracle to determine how to allocate the rewards of the locked reward pool that have accrued: Firstly, the attribution oracle determines what portion of the reward pool to mint, in order to derive the minted CST, R_{mCST}; secondly, after the reward pool is minted, the smart contract uses the fees oracle function to determine the material fees that have accrued and disburses CST from the reward pool to the relevant parties, such as hosting providers, to pay for these fees; finally, the remaining CST left in the reward pool – the actual rewards – are given to recipients and their address balances are updated accordingly.

Parameters on issue, I_1:

- Minting factor of the locked reward pool, σ

Connected Oracles, I_3:

- Attribution oracle, O_a
- Fees oracle, O_f

Outputs:

- Total pool of CST rewards when it is locked, $R_{(t_R)}$
- Subsequent pool of CST rewards when the previous pool is locked, $R_{(t>t_R)}$
- Minted CST after the pool is locked, i.e. $(\sigma \times R_{(t)})$, R_{mCST}

$$C_r((\sigma), (), (O_a, O_f)) = (R_{(t)}, R_{(t>t_R)}, R_{mCST}) \tag{12}$$

2.3.7 Cardstack CST Custodial Smart Contract

The Cardstack CST Custodial Smart Contract is used by the SSC Smart Contract to acquire CST on behalf of Cardstack Users who do not wish to retain CST, but prefer to acquire SSC directly, which can be deposited into their Cardstack Application. The Cardstack CST Custodial Smart Contract sends ethers from the SSC Smart Contract's buy function and uses them to acquire CST at the current rate from either market vendors or the Cardstack Token Smart Contract, where the rate is reported by the asking rate of individual CST holders or the Cardstack Token Smart Contract, respectively. The Cardstack CST Custodial Smart Contract performs the acquisition of CST by using a preference system as to which entity it acquires CST from (in order of descending preference): makers or miners[6]; Foundation (the CST Smart Contract); decentralised exchanges, e.g. another smart token; centralised exchanges. The acquired CST are then sent to the Cardstack CST->SSC Exchange Smart Contract, which captures the CST for the reward pool and issues the Cardstack applications SSC.

Connected Oracles, I_2:

- Exchange oracle, O_e

$$C_{cus}((), (), (O_e)) = (\dots) \tag{13}$$

[6]Makers are the contributors of open-source software who are inactive in securing the network; miners are participating parties who are actively securing the network. Within the Cardstack philosophy, makers and miners are treated similarly in terms of importance.

2.3.8 Cardstack Team Smart Contract

The Cardstack Team Smart Contract is used to distribute CST rewards amongst the maintainers of a Cardstack Application and is instantiated from the Cardstack Application itself. The Cardstack Reward Smart Contract disseminates funds, as usual, to the Cardstack Application Smart Contract's address, a^i (the team). Upon receiving rewards from the Cardstack Reward Smart Contract, the Cardstack Application Smart Contract evokes the Cardstack Team Smart Contract to disseminate the funds to the addresses of all Cardstack Maintainers, $\mathbf{a^i_m}$. The percentage of the reward each Cardstack Maintainer deserves is determined by the weights, $\mathbf{w^i_m}$, which should be encoded in the terms of the agreement, T, that were set in the Cardstack Application Smart Contract.

Parameters on issue, I_1:

- Terms of the agreement, T^i

Connected Contracts, I_2:

- Cardstack Application Smart Contract, C_a
- Cardstack Reward Smart Contract, C_r

Outputs:

- Addresses of the maintainers, $\mathbf{a^i_m}$
- Weights of each maintainer, $\mathbf{w^i_m}$

$$C_T((T^i),(C_a,C_r),()) = (\mathbf{a^i_m},\mathbf{w^i_m}) \tag{14}$$

2.3.9 SSC Top-Off Smart Contract

The SSC Top-Off Smart Contract is owned by the Cardstack User and is used to replenish the balance of SSC, B^i_{SSC}, of the associated Cardstack Application automatically. The Cardstack User adds SSC to the SSC Top-Off Smart Contract as a reserve, BT^i_{SSC}, to be used for refilling the depleted balance. When the Cardstack Application Smart Contract emits the event that the funds are low, the SSC Top-Off Smart Contract replenishes the Cardstack Application Smart Contract with enough funds, such that the minimum SSC balance in the general terms, B_{minSSC}, and the targeted balance, TB_{SSC}, are met.

Parameters on issue, I_1:

- Target balance of SSC, TB^i_{SSC}
- Initial balance of SSC in the SSC Top-Off Smart Contract, BT^i_{SSC}

Connected Contracts, I_3:

- Cardstack Application Smart Contract, C_a

Outputs:

- Balance of SSC in the SSC Top-Off Smart Contract, BT^i_{SSC}

$$C_{top}((TB^i_{SSC}),(),(C_a)) = (BT^i_{SSC},\dots) \tag{15}$$

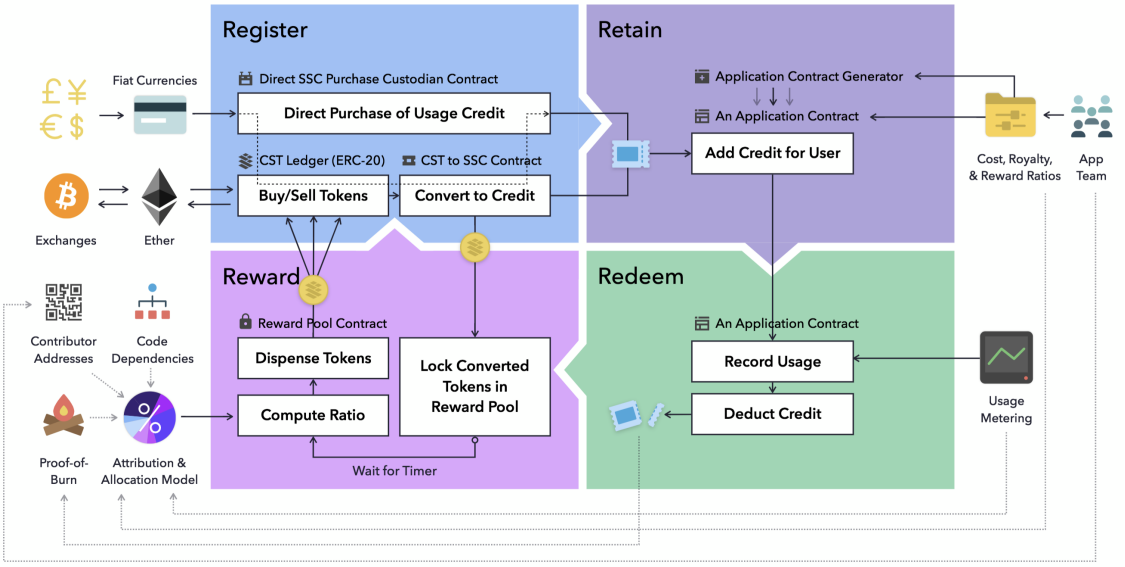

Figure 1: Visualisation of the 4 Rs

3 Token Mechanisms

3.1 The 4 Rs

This section broadly explains the token mechanism. The 4 key steps in the token mechanism are the 4 Rs: Register, Retain, Redeem, Reward. *Register* the Cardstack Application, specifying and agreeing to any terms of the contract; *Retain* SSC in exchange for CST, which enables the redemption of services; *Redeem* services by using the retained SSC; Obtain a *Reward* based on the total amount of SSC redeemed. The 4Rs illustrate the main process of the token mechanism, wherein each step consists of more detailed interactions between oracle functions and contracts – see section 3.2.2. Figure 1 offers a visual aid for imagining the token mechanism. The 4 Rs complement the notion of the 4-edge concept in the whitepaper [7].

3.2 Token Cycle

In this section, we use pseudocode to formalise actions of contracts and oracles in the 4R process.

3.2.1 Notation of pseudocode

Below are the key functions and their definitions:

- CREATE: A contract is created.
- EXECUTE: Transactions are executed on a contract to alter its state or an oracle function is run to return a reported value.
- CALL: Present information inside a contract or the last valid value reported by the oracle is read.
- TRIGGER: An event is triggered within a contract.
- REPORT: A value is reported by an oracle function, which is triggered by a heartbeat function.

All these functions require ether (or gas) provided by the externally controlled Ethereum account, unless the execution occurs off-chain.

[7] Software engineering as part of developing side chains isolates the actions of retaining and redeeming SSC, moving them to an alternative blockchain – this will be discussed in a future paper.

3.2.2 Cardstack Ecosystem Setup

Below is the sequence for the creation of smart contracts by the Cardstack Foundation itself.

1. CREATE C_c (mint 10 billion CST)
2. CREATE C_s
3. CREATE C_e
4. CREATE C_g
5. CREATE C_r
6. CREATE C_{cus}
7. CREATE C_{top}

3.2.3 Register

1. Create application
   (a) As Cardstack Maintainer, C_g CREATE C_a: Instantiate Cardstack Application from Cardstack Application Genesis Smart Contract.
   (b) C_a CREATE C_T (optional): Instantiate Cardstack Team Smart Contract from Cardstack Application Smart Contract.
   (c) EXECUTE C_g: Allocate A_{CST} of CST, as part of network fees paid upon registration, to Cardstack Application Genesis Smart Contract. Increase balance of toll pool, B_{fCST}.
2. Exchange ether for CST
   a1) As Cardstack User, EXECUTE C_c: Exchange ether for CST from CST Smart Contract at rate CP_{Es}.

   OR

   a2) As Cardstack User: Exchange ether for CST from market vendor at available rate.
3. Change Github address or npm address
   (a) As Cardstack Maintainer, EXECUTE C_a: Change npm or Github address, $\bar{a}^i$.

3.2.4 Retain

Consider application address a^i.

1. Exchange CST for SSC
   (a) As Cardstack User, EXECUTE C_e: Exchange CST for SSC at rate CP_{SSCb}.
   (b) C_e CALL O_e: C_e reads last market exchange rate (P_{Es}, P_{Eb}, P_{USs}, P_{USb}) from O_e. Cardstack CST->SSC Exchange Smart Contract sets rates (CP_{SSC_s}, CP_{SSC_b}) based on a function of market exchange rates.
   (c) C_e EXECUTE C_s: Amount A_{SSC} of SSC is issued to user by conversion A_{CST} of CST with rate CP_{SSC_b}.
   (d) C_e EXECUTE C_r: A_{CST} of CST retained in transaction by Cardstack CST->SSC Exchange Smart Contract is allocated to reward pool and balance, R_t, is

increased.

(e) As Cardstack User, EXECUTE C_a: User retains A_{SSC} of SSC within Cardstack Application, C_a. Increase balance of SSC in Cardstack Application, B^i_{SSC}.

2. Exchange ether for SSC

(a) As Cardstack User, EXECUTE C_s: Send A_E of ethers to SSC Contract's ether pool. Increase balance of SSC contract's ether pool, B_E.

(b) C_s EXECUTE C_{cus}: Trigger Cardstack CST Custodial Smart Contract.

(c) C_{cus} EXECUTE C_c (using C_s buy function): Acquire A_{CST} of CST with A_E of ether at disclosed contract rate, CP_{Es}.

(d) C_{cus} EXECUTE C_e: Exchange A_{CST} of CST for A_{SSC} of SSC at rate CP_{SSCb}.

(e) C_e EXECUTE C_s: A_{SSC} of SSC is issued to user.

(f) C_e EXECUTE C_r: A_{CST} of CST retained in transaction is allocated to reward pool and balance, R_t, is increased.

(g) As Cardstack User, EXECUTE C_a: User retains A_{SSC} of SSC within Cardstack Application. Increase balance of SSC, B^i_{SSC}, in Cardstack Application.

3. Adjust CST Smart Contract rates based on exchange oracle (similar adjustments occur for Cardstack CST->SSC Exchange Smart Contract)

(a) REPORT f_λ TRIGGER C_c: At t_λ, heartbeat function triggers CST Smart Contract.

(b) C_c CALL O_e: CST Smart Contract reads exchange rate, (P_{Es}, P_{Eb}).

c1) EXECUTE C_c: Contract rates, (CP_{Eb}, CP_{Es}), are switched to follow (P_{Eb},P_{Es}), i.e. $CP_{Eb} = P_{Eb}$, $CP_{Es} = P_{Es}$.

OR

c2) EXECUTE C_c: Contract rates, (CP_{Eb}, CP_{Es}), are switched to follow (P_{Eb},P_{Es}) based on some formula, i.e. $CP_{Eb} = g(P_{Eb})$, $CP_{Es} = g(P_{Es})$, where g is a formula function.

Note: The CST contract rates can be chosen to be uninfluenced by the market exchange oracle at the discretion of the Cardstack Foundation. A participant can expect that the rates in the CST Smart Contract to acquire CST are higher than the market rates, which is a means to avoid undercutting the market.

3.2.5 Redeem

Consider application address a^i.

1. Redeem service using SSC

(a) REPORT f_λ TRIGGER C_a: At t_λ, heartbeat function executes "heartbeat" and triggers event on Cardstack Application Smart Contract.

(b) C_a CALL O_m: Cardstack Application Smart Contract reads amount of SSC that should be burned, N^i_{SSC}, since $t_{\lambda-1}$, as reported by metering oracle.

(c) EXECUTE C_a: N^i_{SSC} is burned and B^i_{SSC} is reduced. Burn signal is recorded in history of burn signals of amount N^i_{bSSC}.

(d) C_a EXECUTE C_s: Batches of SSC are expired if expiry block height period, T_e, is exceeded; otherwise, they are removed from SSC Smart Contract after amount N^i_{SSC} has been accumulated to redeem services. (This gives us a way to check the expiry of batches of SSC that are earmarked by the time they are added into

the ledger, without expending more gas than we do when this is initiated from the heartbeat function of the metering oracle.)

2. When SSC is low:

   (a) (If $B^i_{SSC} < B_{minSSC}$ OR $B^i_{SSC} < TB^i_{SSC}$) TRIGGER C_a: When balance of SSC is below minimum of SSC required or below target balance in SSC Top-Off Smart Contract, trigger event in Cardstack Application Smart Contract. Assume A_{SSC} is the required amount of SSC to be refilled in B^i_{SSC} to meet conditions.

   b1) If SSC Top-Off Smart Contract is engaged,

      i. C_a EXCUTE C_{top}: A_{SSC} of SSC is taken from balance stored in SSC pool of SSC Top-Off Smart Contract, BT^i_{SSC}. BT^i_{SSC} is deducted.

      ii. C_{top} EXECUTE C_a: A_{SSC} is used to in SSC Top-Off Smart Contract to replenish balance of Cardstack Application Smart Contract, B^i_{SSC}.

   b2) If done manually,

      i. As Cardstack User, EXECUTE C_e: Acquire A_{SSC} of SSC in exchange for A_{CST} of CST, either at rate CP_{SSCs} or from market vendor at available rate. Note: The user can use the Cardstack CST Custodial Smart Contract to get SSC too.

      ii. As Cardstack User, EXECUTE C_a: User retains A_{SSC} of SSC within Cardstack Application Smart Contract. Increase balance of SSC, B^i_{SSC}.

3.2.6 Reward

1. Compute weights

   (a) (When previous reward pool distribution is finished) TRIGGER C_r: At t_R, lock current reward pool at balance R_{t_R} and open next reward pool with balance $R_{t>t_R}$.

   (b) C_r CALL O_f: Compute last reported unpaid fees (t_f), UF.

   (c) EXECUTE C_r: Take UF from reward pool and pay all bills, leaving new reward figure, R_{t_R} .

   (d) C_r EXECUTE O_a (including CALLs to different smart contracts): Run attribution model:

      i. O_a CALL ALL C_a: Find burn signals, $\mathbf{N_{bSSC}}$, in each Cardstack Application Smart Contract. Additionally, find weights of applications' first-level dependencies, $\mathbf{U}$. Read information of Github profile and npm address and obtain package dependency graph, $\bar{\mathbf{a}}$.

      ii. O_a CALL C_s: Read expired SSC burn signals, $\mathbf{B_{eSSC}}$, and sum all up to get B_{eSSC}.

      iii. O_a CALL C_e: Attribution oracle finds value of B_{eSSC} in CST, using rate CP_{SSCs}, calling it B_{eCST}.

      iv. O_a EXECUTE C_r: Send value B_{eCST} to Cardstack Reward Smart Contract.

      v. EXECUTE C_r: Subtract B_{eCST} from reward pool, R_{t_R}, and multiply by minting factor, σ, to obtain R_{mCST}, i.e. $R_{mCST} = \sigma \times (R_{t_R} - B_{eCST})$.

      vi. O_a CALL C_r (C_r sends value): Read size of reward pool, R_{t_R}, and minted CST, R_{mCST}.

      vii. EXECUTE O_a: Carry out proportional attribution computation and application of diversity function. Filter addresses to obtain reward-receiving addresses only, $\mathbf{ra}$.

   (e) O_a EXECUTE C_r: Cardstack Reward Smart Contract accepts reward-receiving addresses, $\mathbf{ra}$, and set of weights corresponding to each address, $\mathbf{w}$.

(f) C_r EXECUTE C_c: Send message to CST Smart Contract to start working on reward cycle.

2. Compute and distribute rewards to reward-receiving addresses (only when reward pool is locked and attribution oracle has been computed)

   (a) REPORT f_λ TRIGGER C_c: Mining function of CST Smart Contract triggers execution, using gas collected through transactions from mining function in CST Smart Contract.

   b1) C_c EXECUTE C_r: Compute size of reward assigned to each address, e.g. $w^i(R_{t_R} + R_{mCST})$ is sent to ra^i.

   OR

   b2) C_c EXECUTE C_r: CST Smart Contract executes a portion of distribution of rewards.

   Repeat until distribution of locked reward pool is completed.