# RIF Name Service

Specification

v1.31-en

# Index

# Preamble

One of the pillars of the World Wide Web, is the Domain Name System (DNS). This system is responsible for creating a mapping between human-readable names and numeric IP addresses. The Internet Corporation for Assigned Names and Numbers (ICANN) is a corporation responsible for coordinating the maintenance and procedures of several databases related to the namespaces and numerical spaces of the Internet, ensuring the network's operation. ICANN performs the actual technical maintenance work of DNS root zone registries.

These services are a central point of trust and failure[1][2]; they can be taken offline by distributed denial-of-service attacks (DDoS) and mappings for domains can be changed by either forcing changes to the DNS servers or by spoofing replies from them. In addition, there are some security concerns such as Internet Service Providers (ISP) being capable of censoring names without easy detection.

One of the goals of RIF Name Service (RNS) is to become a decentralized and secure DNS-like system. RNS works over the RSK Blockchain, thereby inheriting its decentralized nature and security.

An obstacle to cryptocurrency adoption is the difficulty of dealing with addresses. A Bitcoin address has the following aspect: "06f1b66ffe49df7fce684df16c62f59dc9adbd3f", which is notoriously error-prone when a user attempts to type it. Such a long sequence of characters is also difficult to remember, which renders frequent cryptocurrency adoption impractical.

In conclusion, RNS is a decentralized service that gives users a human-readable domain or alias that refers to different resources (e.g., RSK or Swarm addresses). An advantage of using human-readable addresses is a reduction in the apparent complexity of blockchain technology for the end-user making it easier to use.

Domains and subdomains are bought and sold in an open market. The mechanism to obtain a domain for the first time is through a blind Vickrey auction [3] bidding with RIF Tokens. The practice has shown that human psychological quirks, and not just supply and demand, drive auctions. Vickrey auction mechanism reduces the likelihood that a bidder will overpay for an item, and increases the likelihood that the seller obtains the most she can get for it. Once a user wins an auction, ownership of the domain is assigned to her and an annual rent is paid in order to retain such ownership. Details of how this rent is computed will be explained in the "Registrar" technical section of this document.

The initial guidelines provided in this protocol may be subject to further changes, as the ideas and architecture are discussed and improved by the ecosystem in the future.

# The RIF Ecosystem

## RIF Token

In the RNS token economy, the main function of the RIF token is to prevent unused storage, due to unused domains, or to prevent name squatting. The domain owner should have incentives to forfeit any unused domain's ownership. To achieve this, the domain owner locks RIF tokens which are refunded once the domain is released. In addition, any fee or penalization payment must be performed using RIF Tokens, which will be allocated to the Network Resource Pool. The Network Resources Pool exists to subsidize certain network expenses that will be necessary in the early stages of adoption of the RIF Name Service. The end goal is to enable decentralized governance of the funds.

## RIF Open Standard (RIFOS)

RIF Explorer, implemented by RIF Labs, is an abstraction layer that allows each RIF Service to be decoupled from a particular implementation. The particular implementations are known to the platform as service providers. This decoupling enables each service to evolve as new technology enhancements become available.

In order to support different service providers for each service of the RIF Platform, RNS is used as a service discovery mechanism. This will allow users and developers to find out what service providers are available and how to communicate with them. To understand more about the RIF Explorer, read its corresponding white paper [4].

# Introduction

The following section provides a general description of the RNS components. For more details, read the Technical Section.

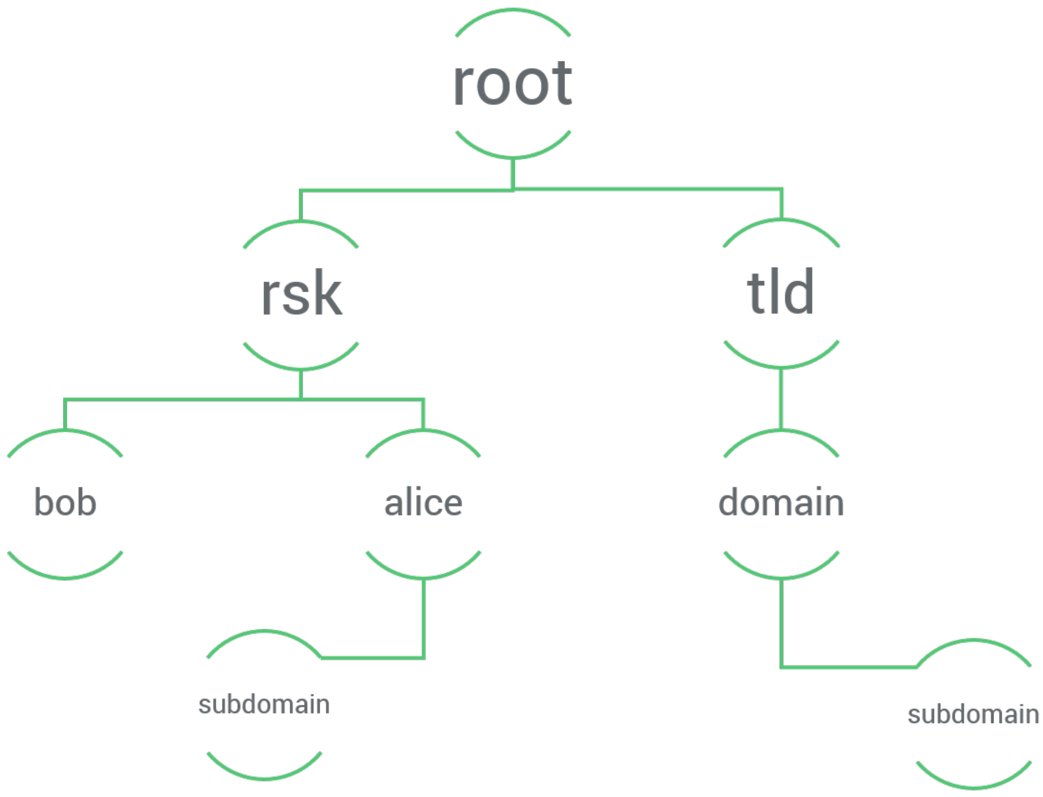
## Components overview

The RNS architecture is based on the Ethereum Name Service (ENS) described on EIP-137 [5]. It is divided into three main components: Registry, Resolver, and Registrar.

### Node

In RNS, the top-level name is defined as "*.rsk*". Second-level names are called domains. In addition, a domain can reference to different resources through subdomains. A node is a part of a subdomain hierarchy-tree. A path to the root in this tree is a domain name.

This path is split into components by a dot ("."). Each node stores a hash of the name string of an intermediate component, computed by the name hash algorithm explained above.



### Name Format

The RNS name format must conform to the following syntax: "subdomain.domain.rsk". In short, the name consists of a series of dot-separated labels, each one consisting of a subdomain tree-level. In addition, the subdomain name must conform to the rules explained in the Name Format technical section.

## Resolver

Resolver contracts are responsible for the resolution of a resource name. A resolver has many user-defined functions and each function enables a different resource type to be fetched on the same node.

## Registry

The registry contract provides a simple mapping between a domain and its resolver. Everything related to domain ownership is managed in this contract, including ownership transfer and sub-domain creation.

## Registrar

The registrar is responsible for the RNS governance. In addition, it is responsible for registering the name of a domain for a user, and is the only entity capable of updating the

RNS Registry. In the event that the domain is migrated to another registrar (migration process is explained below), it can delegate ownership of subdomains to other registrars.

As explained before, at the initial stages of RNS only one top-level domain (TLD) will be available: ".rsk". By reducing the number of top-level domains, we can focus on the registration of second-level domains, and postpone the discussion about top-level overlaps with the standard DNS system and other name services.

To prevent domain spamming or squatting, the registrar will manage the auctioning process. Since RNS payments are denominated in RIF tokens, the registrar interacts with the ERC 677 RIF Token contract in order to make payments between accounts. Initially it will only handle ".rsk" TLD, and subdomains of any char-length. This will be restricted from the RNS root node, which is controlled by a multi-signature contract. This multi-signature contract will be initially controlled by RSK Labs. during the beta period.

### Deed

In order to prevent unnecessary used storage due to unused domains or prevent name squatting, the domain owner should have incentives to forfeit their ownership of them. To achieve this, the domain owner locks tokens which are refunded once the domain is released.

## Main use cases

### Register a domain

There are two ways users can get a domain. The first is to open an auction through the registrar contract for the desired domain. For example, if ".rsk" is the TLD and the user Alice wants the domain "alice.rsk," she can open an auction for this domain, make a bid, and if it is the highest, she will become the new owner of "alice.rsk." A second way is, if Bob is the owner of "bob.rsk" and Alice wants the subdomain "subdomain.bob.rsk," Bob can delegate the subdomain ownership to Alice without an auction process. Once a user gets a domain, she should define in the domain's registry contract the resolver that will perform the resolution between the new domain and the desired resource. If a user doesn't set a resolver, a default one is set. This default resolver is the new owned domain's parent's resolver. Then, the new domain's owner should create a resolver entry with her new domain. For example, if Alice doesn't set a resolver on "subdomain.bob.rsk" registry entry, "bob.rsk" resolver is set.

### Resolve a domain

Domain resolution is a process that verifies whether a domain exists and then returns information associated with its registry entry. This resolution can be used in wallets, exchanges or dApps, to handle domains without any knowledge about their respective addresses. To do this, first, let's assume the domain "bob.rsk" exists. To resolve "bob.rsk" domain, a user must use the name-hash algorithm explained below which returns a node (e.g. 0x231..de3). After that, given that node, the user must look for the RNS Registry entry associated with it. This entry contains the resolver that given the node returns the desired resource.

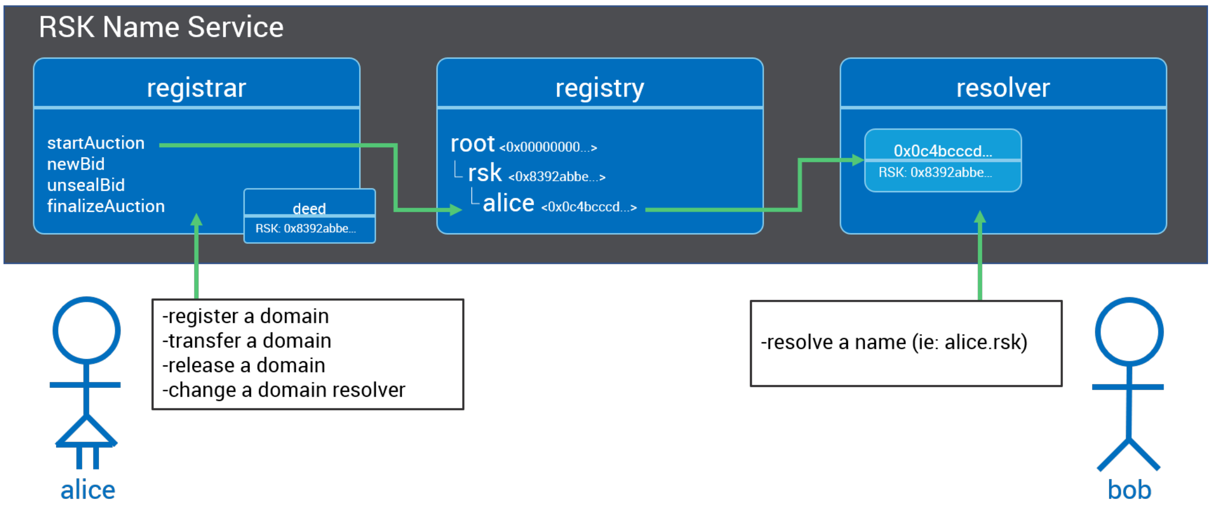
## Technical Overview

RSK Labs deploys a registry contract that handles the mapping between a domain name and its owner. Each registry entry refers to a resolver which handles the resolution between the name domain and the desired resource.

Next, the registrar contract that manages the auction and supply of domains is deployed. For each bid, the registrar creates a deed contract and transfers the user's bid amount to the deed. The auction winner must lock the deed's balance and exchange the deed for the domain ownership. Then, the system registers the auction winner in the registry as the domain owner and she may set her own resolver.

In addition, the owner can delegate subdomains using the registry contract without an auction process.

Every name owner has a deed for each name and must pay an annual rent for each deed. The reason that registry rents are charged is to prevent domain squatting and domain spamming in the registry contract storage. If the rent is unpaid, the domain ownership is cancelled and the state of the domain in the registrar contract becomes open to new auctions (the open state).



## Components

### Registry

The guidelines and interface for the Registry are described in RNSIP01[6].

#### Name format

RNS names must conform to the following syntax:

```
<domain> ::= <label> | <domain> "." <label>
<label> ::= any valid string label per [7]
```

In short, names consist of a series of dot-separated labels. Each label must be a valid normalized label as described in UTS46 [7] with the options transitional=false and use STD3AsciiRules=true. For JavaScript implementations, a library [8] is available that normalizes and checks names.

Note that while upper and lower-case letters are allowed in names, the UTS46 normalization process case-folds labels before hashing them, so two names with different cases but identical spelling will produce the same name-hash.

Labels and domains may be of any length, but for compatibility with legacy DNS, it is recommended that labels be restricted to no more than 64 characters each, and complete RNS names to no more than 255 characters. For the same reason, it is recommended that labels do not start or end with hyphens, or start with digits.

### Name hash algorithm

RNS uses the name-hash algorithm. This algorithm recursively hashes components of the name, producing a unique, fixed-length string for any valid input domain

The output of name-hash is referred to as a 'node.'

Pseudocode for the name-hash algorithm is as follows:

```
def namehash(name):
  if name == '':
    return '\0' * 32
  else:
    label, _, remainder = name.partition('.')
    return sha3(namehash(remainder) + sha3(label))
```

## Resolver

The resolver is an interface. A user can use the public resolver contract provided by RSK or implement their own resolver contract. If a user doesn’t set her own resolver for her registry entry, the parent’s domain resolver will be used. Then the user should register in the parent’s domain resolver the resolution information between the domain name and the desired resource. Like with the registry, the resolver interface and specification are described in the RNSIP01 [6].

## Registrar

### Registrar Interface

***constructor(RNS _rns, bytes32 _rootNode, uint _startDate, ERC677 tokcAddr)***

- The constructor receives an RNS Registry, a root node that the registrar belongs to, as well as an ERC 677 token contract to use for RIF Payments.

***startAuction(bytes32 _hash) public***

- Changes the state of a hash from open to auction.

***startAuctions(bytes32[] _hashes) public***

- Allows anyone to start an auction for a number of hashes. This method can be used to prevent an attacker bidding blindly for auctions. In this case, some of the submitted hashes are dummy hashes, while the sender is only interested in bidding for one. This will increase the cost for an attacker to simply bid blindly on all new auctions. Dummy auctions opened but not bid on are closed after a week.

***newBid(bytes32 sealedBid, uint tokenQuantity) public***

- Bids are created by sending a message to the main contract with a sealedBid hash (created with shaBid function) and a number of tokens. The hash contains information about the bid, including the bid-on name-hash, the bid value, and a random salt. Bids are not tied to any auction until they are revealed. The value of the bid itself can be masqueraded by sending more than the actual bid value. Once the auction period ends, this is followed by a 48-hour reveal period. If a bid is revealed after this period, it could be penalized with the offered tokens. Since this is an auction, it is expected that most public hashes, like known domains and common dictionary words, will have multiple bidders pushing the price up. Finally, a deed is created with a number of tokens and a contract for managing them.

***newBidWithToken(bytes32 sealedBid, uint tokenQuantity,) public***

- Equivalent to newBid. It is useful for calls from ERC 677 contracts.

***startAuctionsAndBid(bytes32[] hashes, bytes32 sealedBid, uint tokenQuantity) public payable***

- A utility function allowing a call to startAuctions followed by newBid in a single transaction.

***unsealBid(bytes32 _hash, uint _value, bytes32 _salt) public***

- Once the bidding period is completed, there is a reveal period during which the proof of ownership of a bid is submitted. The registrar hashes these parameters using the shaBid() function to verify that they match a pre-existing sealed bid. If the unsealedBid is the new best bid, the old best bid is returned to its bidder.

***cancelBid(address bidder, bytes32 seal) public***

- Cancels an unrevealed bid according to the rules described in the refund schedule below.

***finalizeAuction(bytes32 _hash) public onlyOwner(_hash)***

- After the reveal period is finished, this function must be called to finalize the auction. After the auction is closed, the RNS Registry is updated with the highest bidder as the new owner for the auctioned name.

***transfer(bytes32 _hash, address newOwner) public onlyOwner(_hash)***

- Updates the RNS Registry, transferring the ownership of a label hash to a new owner.

***releaseDeed(bytes32 _hash) public onlyOwner(_hash)***

- Nine months after the deed is created, the owner of a name can call this method to forfeit the name and have a portion of their deed funds returned to them.

***eraseNode(bytes32[] labels)***

- Allows anyone to delete the owner and resolver records for a subdomain of a name that is not currently owned in the registrar. For instance, to zero my.example.rsk on a registrar that owns ".rsk", pass an array containing [sha3('my'), sha3('example')].

***transferRegistrars(bytes32 _hash)***

- If this registrar is no longer the owner of the root node in the RNS, this function will transfer the deed to the current owner, which should be a new registrar. This function throws an error if this registrar still owns its root node.

***shaBid(bytes32 hash, address owner, uint value, bytes32 salt) public pure returns (bytes32)***

- Hash the values required for a secret bid.

***payRent(bytes32 _hash) public***

- Pay the annual rent for a domain.

***payRentWithTokens(bytes32 _hash) public***

- Equivalent to payRent. It is useful for calls from ERC 677 contracts.

***acceptedRegistrarTransfer(bytes32 _hash, DeedWithTokens _deed, uint _registrationDate) public pure***

- Accept node transfers and change their state for a registrar migration.

***tokenFallback(address _from, uint _value, bytes _data) public***

- Function needed to make transfers with ERC 677.

### Auction Process

The Vickrey auction is a four-step process:

- **Open**: domain's default state.
- **Auction:** Auction started. There is a 72-hour period where users can submit their sealed bids. The sealed bids can be obtained through *shaBid(bytes32 hash, address owner, uint value, bytes32 salt).*

- **Reveal:** After the auction, there is a 48-hour reveal period. Each bidder reveals his/her bid and the registrar updates the auction accordingly. If no bids are revealed, the state returns to open.
- **Owned:** When the reveal period ends, the winner must submit a transaction to finalize the reveal time using the *finalizeAuction* method. This finalizes the auction and records the winner as the owner of the auctions name hash.

Refund and penalization schedule

| **Bid Result** | **Description** | **Payments involved** |
|---|---|---|
| User wins the auction | If a user wins the auction, the second highest bid amount will be locked in her deed (TL) and the difference between that amount and the highest bid is refunded. A percentage of this locked amount minus the annual rent amount (Y) will be paid as a fee (F) when the auction finished. | T2: Second highest value<br>Y: Year rent price<br>TL: Amount Locked in Deed<br>F: Fee<br>T = (T2 - Y)<br>F= T*0.2<br>TL = T*0.8 |
| User loses the auction | If a user loses the auction because her offer is not the highest, a percentage of the amount locked (TL) in her deed will be paid as a fee (F) | TL: Amount Locked in Deed<br>F: Fee<br>F = 0.05*TL |

A period to reveal the blind bids (reveal period.) If a bidder cancels the bid before the reveal period starts, 99.5% of the bid tokens are refunded. Then, if the bidder reveals the bid before the reveal period starts, the transaction is reverted. When this period expires, each bid is settled. Let's say T is the winning bid amount, T2 is the second highest bid amount, and V is a bid revealed after the reveal period has concluded, the payments and refunds are governed by the following conditions:

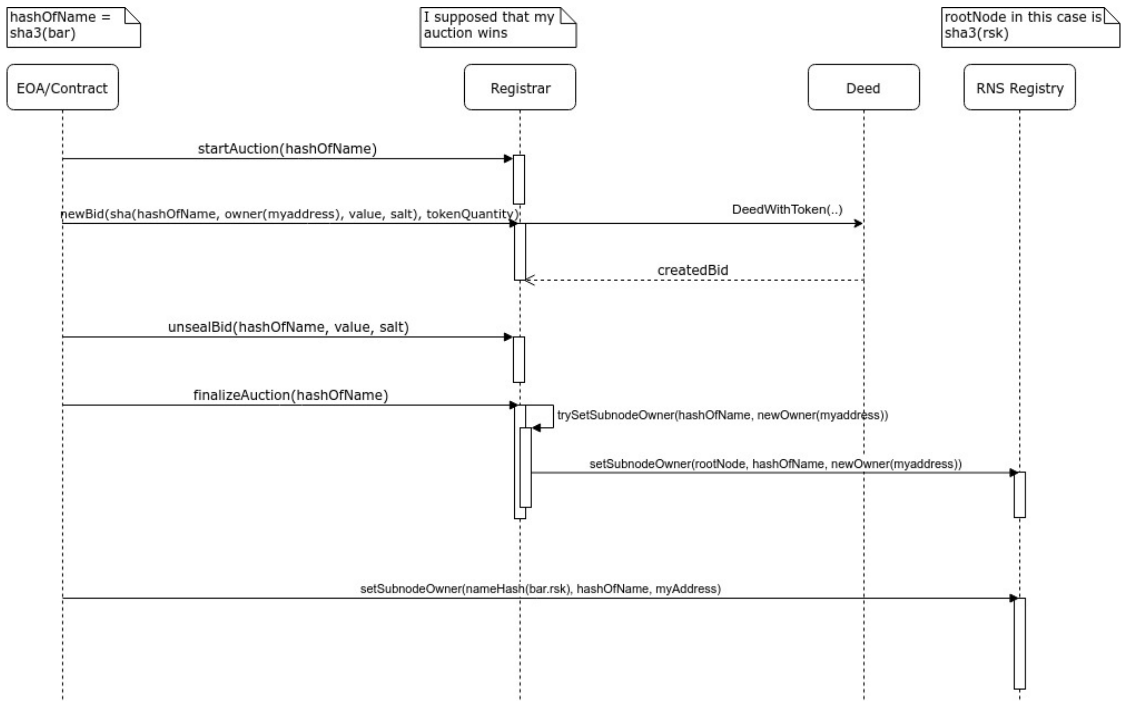
| **Condition** | **Description** | **Payments involved** |
|---|---|---|
| If V > T | If it was revealed in time, it would have won | V*0.2 will paid as a fee and the rest refunded |
| If T > V > T2 | If it was revealed in time, it would have been the second highest value | V - T2 will paid as a fee and the rest refunded |

| | | |
|---|---|---|
| Otherwise | It wasn’t revealed and is less than the second highest value | V*0.05 will paid as a fee and the rest refunded |

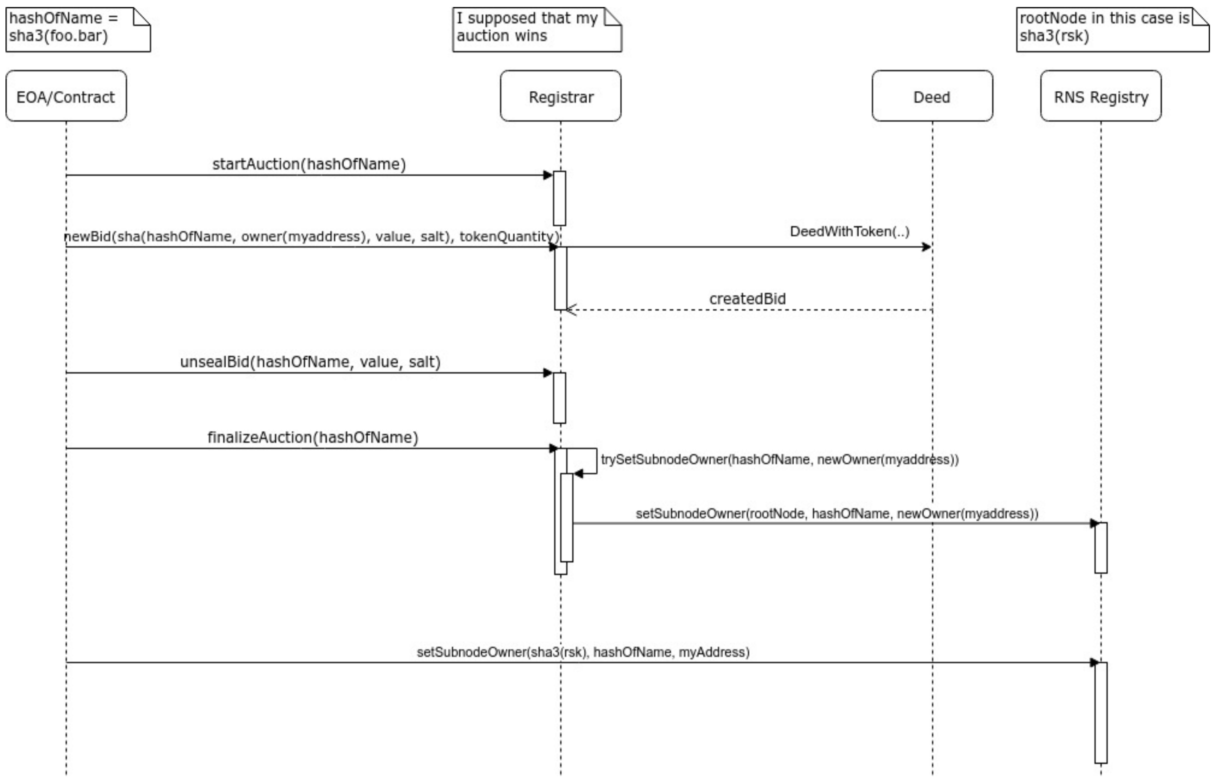
After the reveal period ends, the user has 15 days to reveal the bid. If the user does not reveal it still, the full locked tokens on the deed will be sent to the RNS Network Resources Pool, which is done to prevent RIF Tokens from getting locked forever.

## Uniqueness of Domains

Supposing Alice gets the ownership of the domain “bob.alice.rsk” and delegates ownership of subdomain “bob.alice.rsk” to another user Bob. The sequence to get the ownership of a subdomain would be the following:


hashOfName = sha3(bar)
I supposed that my auction wins
rootNode in this case is sha3(rsk)
EOA/Contract
Registrar
Deed
RNS Registry
startAuction(hashOfName)
newBid(sha(hashOfName, owner(myaddress), value, salt), tokenQuantity)
DeedWithToken(..)
createdBid
unsealBid(hashOfName, value, salt)
finalizeAuction(hashOfName)
trySetSubnodeOwner(hashOfName, newOwner(myaddress))
setSubnodeOwner(rootNode, hashOfName, newOwner(myaddress))
setSubnodeOwner(nameHash(bar.rsk), hashOfName, myAddress)

This interaction may suggest that a malicious user Mallory can start an auction for the domain `sha3(“subdomain.bob”)` in the “.rsk” registrar even if she’s not the rightful owner of “.bob.” This is because auctions are for the name-hash of a domain name and not for the name string. Let’s assume Mallory wants to squat the “subdomain.bob.rsk” domain owned by Bob. The process would be the following:


hashOfName = sha3(foo.bar)
I supposed that my auction wins
rootNode in this case is sha3(rsk)
EOA/Contract
Registrar
Deed
RNS Registry
startAuction(hashOfName)
newBid(sha(hashOfName, owner(myaddress), value, salt), tokenQuantity)
DeedWithToken(..)
createdBid
unsealBid(hashOfName, value, salt)
finalizeAuction(hashOfName)
trySetSubnodeOwner(hashOfName, newOwner(myaddress))
setSubnodeOwner(rootNode, hashOfName, newOwner(myaddress))
setSubnodeOwner(sha3(rsk), hashOfName, myAddress)

Afterward, Mallory will set an entry on the RNS Registry for the sha3((sha3('rsk'), sha3('subdomain.bob')) using his own resolver contract with the intention of redirecting the resolution of "subdomain.bob.rsk" to a different resource. But when a user looks up for the domain name 'subdomain.bob.rsk', the name-hash algorithm (explained above) will resolve sha3(sha3(sha3('rsk'),sha3('bob')), sha3('subdomain')) instead of sha3(sha3('rsk'), sha3('subdomain.bob')). Consequently, the domain name resolved by the name-hash algorithm will be the one defined by Bob.

## Deed

Every RIF token sent to the registrar is stored in a separate contract, called a "deed." Each deed stores the token balance of a specific name hash. The deed is created when a bid is submitted. Then, once the auction is completed and the domain registered, the winning deed and bid will be locked and exchanged for domain ownership. The deed balances pertaining to losing bids are refunded to their rightful owners upon request. Like the registrar, a deed contract knows the RIF Token ERC 677 contract that handles token payments.

A deed for an owned name may be transferred to another account by its owner, thus transferring ownership and control of the name. This is done through the registrar contract.

Nine months after an auction is concluded, the node owner will have the option to pay an annual rent, renewing domain ownership for another year. If the owner doesn't want to pay another annual rent, the owner may opt to relinquish ownership and have the funds frozen in the deed returned to him/her.

To pay the rent of any domain, a user can use the registrar's payRent function. The function requires a RIF token payment. If the owner opts to relinquish ownership, nine months after the deed's creation, she has three months to call releaseDeed and have the locked tokens refunded. After this three-month period, the auction state of that domain will switch to open and the full deed balance will be transferred to the RNS Network Resource Pool.

Deeds for non-winning bids can be closed through various methods, at which time any RIF tokens held will be returned to the bidder.

## ERC 677 Token Contract

The Token RIF is implemented with the ERC 677 Standard. Payments for annual rent or bids on RNS are made with RIF tokens. So, the process to interact with the ERC 677 RIF Token Contract is the following:

- In the `transferAndCall` function with 3 parameters, the signatures must be:
    - newBid: Set *data* parameter the signature 0x1413151f, concatenated with the sealedBid created by the shaBid function.
    - payRent: Set *data* parameter the signature 0xe1ac9915, concatenated with the sha3 of the label for which rent is paid.

## Improvement Proposals

The contracts that belong to RNS architecture can be upgraded to introduce new improvements. These upgrades are provided based on community feedback and RSK Labs's proposals. Every RNS upgrade has backward compatibility, in other words, domain ownership is kept by their owners.

### Subdomain Management

There are two ways a user can acquire a sub-domain of a specified domain. If the domain owner is a registrar, the user can start an auction of any subdomain of that domain. Also, the domain owner can delegate a subdomain to the buyer without going through an auction process by using the setSubnodeOwner function. This last option doesn't generate any incentive to the new owner to delete the registry entry of the subdomain when is not in use anymore, this is because there is no locked value, because there is no deed contract. We are working on a better delegation system and a fair subdomains management procedure.

### New Registry Structure

The registry contract is the only valid one for node ownership. The registry stores all RNS information. This won't scale once storage rent has been implemented. We are researching an alternative registry structure in order to obtain fairer storage rent.

### DNS Domains and Oracles

There will be also the possibility of migrating regular DNS addresses to the RNS. A DNS address owner would be able to claim a domain in the RNS through the use of oracles to verify he/she is the rightful owner of the original domain. In case of collision with the ICANN name domain, an arbitrage system will be used to resolve conflicts. Said arbitrage system could be implemented through oracles, along with other methods.

### Resolver Anonymity

Users may want to hide the address their domain maps to. This can be done with encrypted resources. The owner may transfer the decryption key to a user upon request though an off-chain communication channel. Also, the addresses stored can be stealth, so that the sender must generate a new address for each independent payment.

### Creating a new Top-Level Registrar

RSK Labs provided an initial registrar for TLD (.rsk). In the future, users may be able to create their own TLD deploying their own registrars. The user may enable an auction process (or a different process) to let people acquire subdomains.

## References

[1] M. Ali, R. Shea, J. Nelson, M J. Freedman, "Blockstack: A New Internet for Decentralized Applications" (2017) https://blockstack.org/whitepaper.pdf
[2] "Namecoin FAQS" https://namecoin.org/docs/faq/
[3] "Vickrey Auction" https://en.wikipedia.org/wiki/Vickrey_auction
[4] "RIF Explorer" https://docs.rifos.org/rif-explorer-specification-en.pdf
[5] N. Johnson, "Ethereum Domain Name Service" (2016) https://github.com/ethereum/EIPs/blob/master/EIPS/eip-137.md
[6] J. Len, "Registry and Resolver of RNS" (2018) https://github.com/rnsdomains/RNSIPs/blob/master/IPs/RNSIP01.md
[7] M. Davis, M. Suignard "UTR46" http://unicode.org/reports/tr46/
[8] NPM Library https://www.npmjs.com/package/idna-uts46