RecordsKeeper Documentation

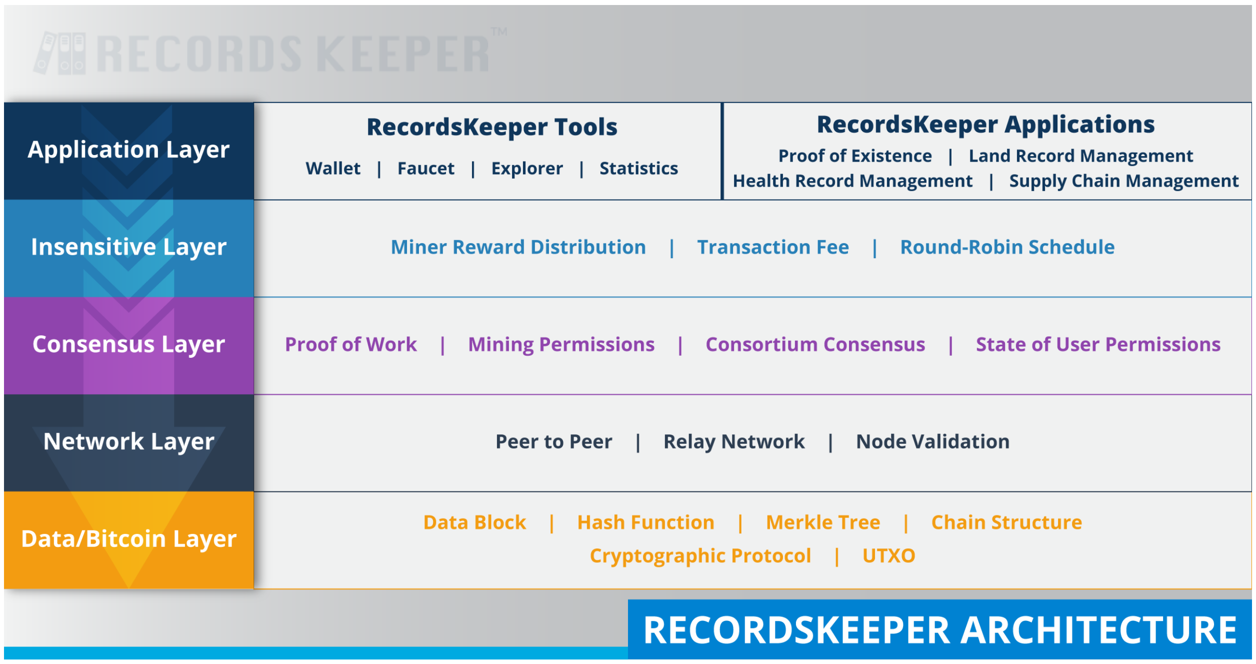
RecordsKeeper

May 27, 2021

Contents

RecordsKeeper's blockchain-centric solution is built to store, share, and manage your information in the safest fashion. Once your data is on the RecordsKeeper blockchain, it cannot be deleted, tampered with, or modified even by the data owner. RecordsKeeper offers a rich set of features including extensive configurability, rapid deployment, permissions management, native assets, and data streams. There are several use cases for RecordsKeeper ranging from KYC verifications, supply chain management, manufacturing, health record management, academic certifications, and employment credentials verification.

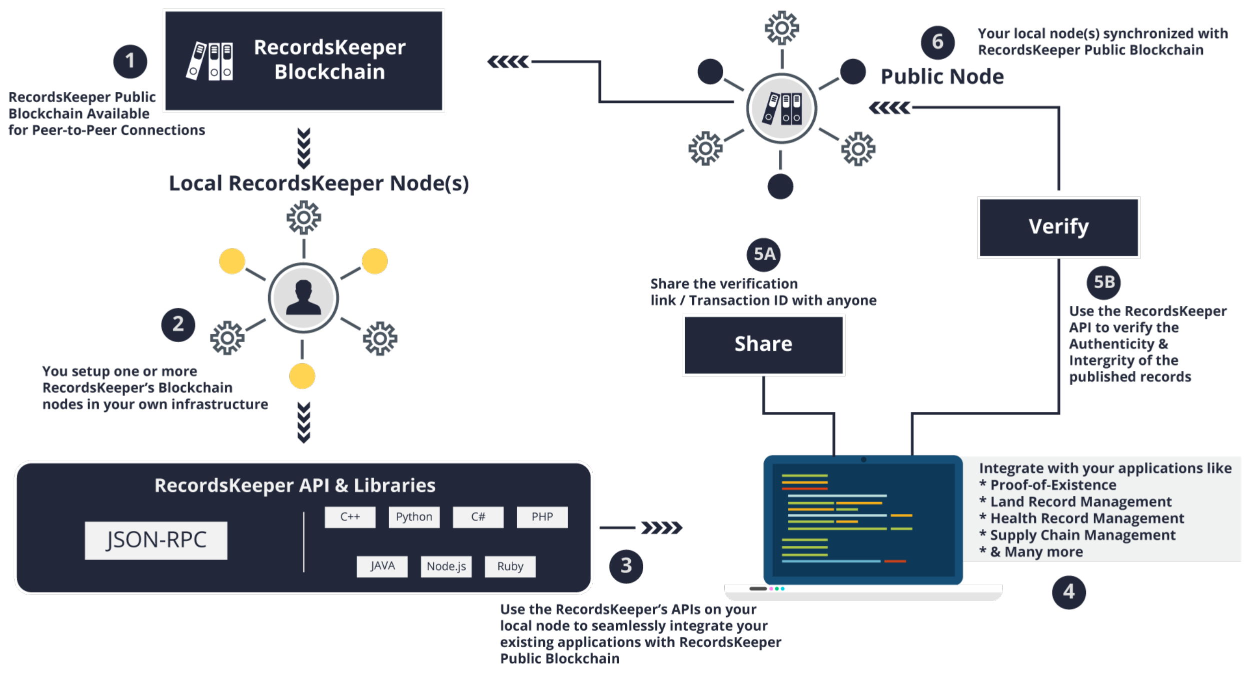
The ecosystem for RecordsKeeper includes different layers which provide the building blocks for the RecordsKeeper blockchain. The different layers, as shown in the diagram below, interact with each other to form the public, minable database on the blockchain.


RECORDS KEEPER™
Application Layer
RecordsKeeper Tools
Wallet | Faucet | Explorer | Statistics
RecordsKeeper Applications
Proof of Existence | Land Record Management
Health Record Management | Supply Chain Management
Insensitive Layer
Miner Reward Distribution | Transaction Fee | Round-Robin Schedule
Consensus Layer
Proof of Work | Mining Permissions | Consortium Consensus | State of User Permissions
Network Layer
Peer to Peer | Relay Network | Node Validation
Data/Bitcoin Layer
Data Block | Hash Function | Merkle Tree | Chain Structure
Cryptographic Protocol | UTXO
RECORDSKEEPER ARCHITECTURE

RecordsKeeper was founded by Toshendra Sharma and Rohendra Singh in November 2016 and includes a team of blockchain developers, and marketing veterans who share a passion for blockchain technology and believe in collective shared information and data. RecordsKeeper is based in Singapore with offices in India. The team at RecordsKeeper has created an open-source, immutable, public database using blockchain technology which is available to millions of users to store and share data, and currently a new block of data is being created every 15 seconds.

Note: The best way to get familiar with RecordsKeeper is to try the Demo . The Demo allows you to create a new wallet and store data on the RecordsKeeper blockchain. It also allows you to retrieve the data and verify it.

Individual users and organizations can create their own private nodes which can act as building blocks for their applications. The standard JSON RPC APIs provided with RecordsKeeper provide seamless integration of applications with the RecordsKeeper blockchain. These applications are synchronised with a RecordsKeeper node, and data can then be viewed and verified by the parties involved or shared globally. Check the diagram below to see how RecordsKeeper blockchain works:


1
RecordsKeeper Blockchain
RecordsKeeper Public Blockchain Available for Peer-to-Peer Connections
Local RecordsKeeper Node(s)
2
You setup one or more RecordsKeeper's Blockchain nodes in your own infrastructure
RecordsKeeper API & Libraries
JSON-RPC
C++
Python
C#
PHP
JAVA
Node.js
Ruby
3
Use the RecordsKeeper's APIs on your local node to seamlessly integrate your existing applications with RecordsKeeper Public Blockchain
4
Integrate with your applications like
* Proof-of-Existence
* Land Record Management
* Health Record Management
* Supply Chain Management
* & Many more
5A
Share the verification link / Transaction ID with anyone
Share
5B
Use the RecordsKeeper API to verify the Authenticity & Intergrity of the published records
Verify
6
Your local node(s) synchronized with RecordsKeeper Public Blockchain
Public Node

CHAPTER 1

Useful Links

- RecordsKeeper
- RecordsKeeper Demo
- RecordsKeeper Wallet
- RecordsKeeper Stats
- RecordsKeeper Miners Information
- Mainnet Explorer
- Testnet Explorer
- Source Code

CHAPTER 2

Language Documentation

On the next pages, we will see how to use RecordsKeeper to store data followed by the basics about RecordsKeeper mining and how to set up your network.

The next section will explain several features of RecordsKeeper. You can implement and deploy your own solutions on the RecordsKeeper blockchain.

The last section will cover important aspects and use cases of RecordsKeeper in depth.

If you still have questions, you can try searching or asking on the RecordsKeeper FAQ, or come to our Telegram channel. Ideas for improving RecordsKeeper or this documentation are always welcome!

CHAPTER 3

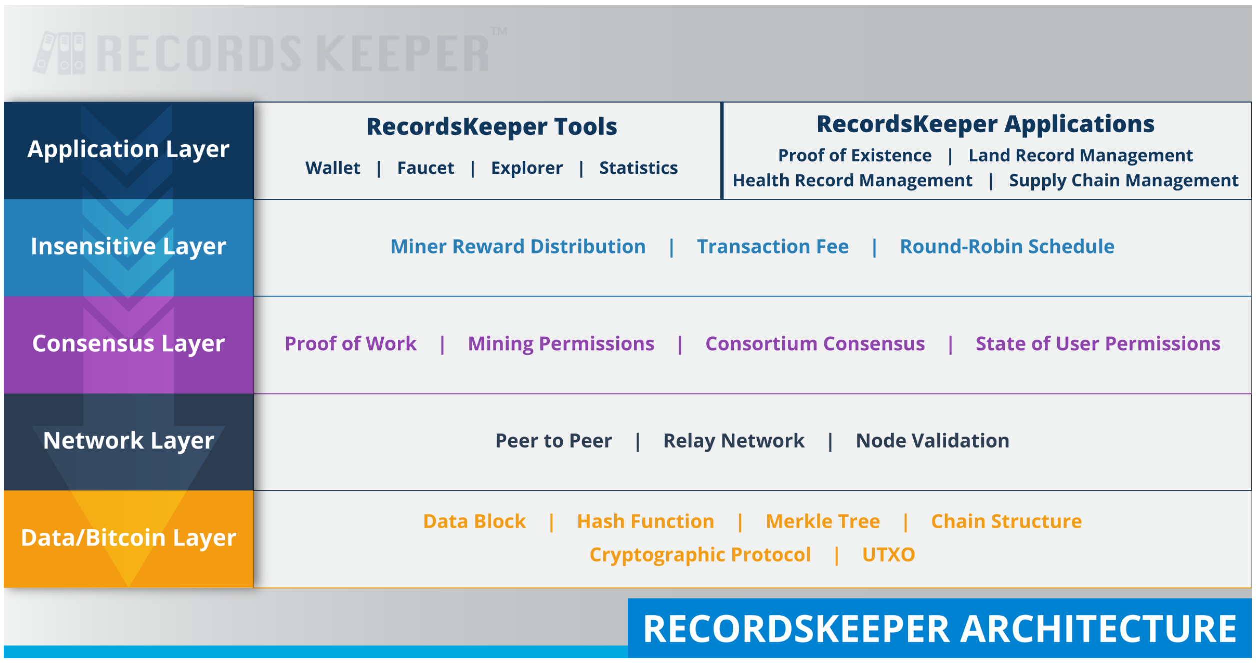
Contents

genindex, modindex, search

3.1 RecordsKeeper Overview

RecordsKeeper's blockchain-centric solution is built to store, share, and manage your information in the safest fashion. Once your data is on the RecordsKeeper blockchain, it cannot be deleted, tampered with, or modified even by the data owner. RecordsKeeper offers a rich set of features including extensive configurability, rapid deployment, permissions management, native assets, and data streams. There are several use cases for RecordsKeeper ranging from KYC verifications, supply chain management, manufacturing, health record management, academic certifications, and employment credentials verification.

The ecosystem for RecordsKeeper includes different layers which provide the building blocks for the RecordsKeeper blockchain. The different layers, as shown in the diagram below, interact with each other to form the public, minable database on the blockchain.


RECORDS KEEPER™
Application Layer
RecordsKeeper Tools
Wallet | Faucet | Explorer | Statistics
RecordsKeeper Applications
Proof of Existence | Land Record Management
Health Record Management | Supply Chain Management
Insensitive Layer
Miner Reward Distribution | Transaction Fee | Round-Robin Schedule
Consensus Layer
Proof of Work | Mining Permissions | Consortium Consensus | State of User Permissions
Network Layer
Peer to Peer | Relay Network | Node Validation
Data/Bitcoin Layer
Data Block | Hash Function | Merkle Tree | Chain Structure
Cryptographic Protocol | UTXO
RECORDSKEEPER ARCHITECTURE

RecordsKeeper was founded by Toshendra Sharma and Rohendra Singh in November 2016 and includes a team of blockchain developers, and marketing veterans who share a passion for blockchain technology and believe in collective shared information and data. RecordsKeeper is based in Singapore with offices in India. The team at RecordsKeeper has created an open-source, immutable, public database using blockchain technology which is available to millions of users to store and share data, and currently a new block of data is being created every 15 seconds.

Note: The best way to get familiar with RecordsKeeper is to try the Demo . The Demo allows you to create a new wallet and store data on the RecordsKeeper blockchain. It also allows you to retrieve the data and verify it.

3.1.1 Useful Links

- RecordsKeeper
- RecordsKeeper Demo
- RecordsKeeper Wallet
- RecordsKeeper Stats
- RecordsKeeper Miners Information
- Mainnet Explorer
- Testnet Explorer
- Source Code

3.1.2 Language Documentation

On the next pages, we will see how to use RecordsKeeper to store data followed by the basics about RecordsKeeper mining and how to set up your network.

The next section will explain several features of RecordsKeeper. You can implement and deploy your own solutions on the RecordsKeeper blockchain.

The last section will cover important aspects and use cases of RecordsKeeper in depth.

If you still have questions, you can try searching or asking on the RecordsKeeper FAQ, or come to our Telegram channel. Ideas for improving RecordsKeeper or this documentation are always welcome!

3.2 Installing the RecordsKeeper Node

3.2.1 Versioning

RecordsKeeper nodes follow semantic versioning, and in addition to releases development builds are made available through source code. The development builds are not guaranteed to be working and despite our best efforts they might contain undocumented and/or broken changes. We recommend using the latest release. Package installers below will use the latest release.

3.2.2 Linux Pre-Installation Requirements (on Ubuntu 14.04 x64)

Install dependencies on your operating system.

Note: You need to install apt-get or any other package manager to install the dependent libraries.

```
sudo apt-get update
sudo apt-get install build-essential libtool autotools-dev automake pkg-config libssl-
↪dev libevent-dev bsdmainutils
sudo apt-get install libboost-all-dev
sudo apt-get install git
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:bitcoin/bitcoin
sudo apt-get update
sudo apt-get install libdb4.8-dev libdb4.8++-dev
```

Building from Source

Clone the Repository

To clone the source code, execute the following command:

```
git clone https://github.com/RecordsKeeper/recordskeeper-core.git
cd recordskeeper-core
```

Compile RecordsKeeper for Ubuntu (64-bit)

This will build rkd, rk-cli and rk-util in the src directory.

```
./autogen.sh
./configure
make
```

3.2.3 Windows Pre-Installation Requirements (on Ubuntu 14.04 x64)

Install dependencies on your operating system.

Note: You need to install apt-get or other package manager to install the dependent libraries.

```
sudo apt-get update
sudo apt-get install build-essential libtool autotools-dev automake pkg-config libssl-
↪dev libevent-dev bsdmainutils
sudo apt-get install g++-mingw-w64-i686 mingw-w64-i686-dev g++-mingw-w64-x86-64 mingw-
↪w64-x86-64-dev curl
sudo apt-get install libboost-system-dev libboost-filesystem-dev libboost-chrono-dev␣
↪libboost-program-options-dev libboost-test-dev libboost-thread-dev
sudo apt-get install git
sudo add-apt-repository ppa:bitcoin/bitcoin
sudo apt-get update
sudo apt-get install libdb4.8-dev libdb4.8++-dev
```

Building from Source

Clone the Repository

To clone the source code, execute the following command:

```
git clone https://github.com/RecordsKeeper/recordskeeper-core.git
cd recordskeeper-core
```

Compile RecordsKeeper for Windows (64-bit)

This will build rkd, rk-cli and rk-util in the src directory.

```
./autogen.sh
cd depends
make HOST=x86_64-w64-mingw32 -j4
cd ..
./configure --prefix=`pwd`/depends/x86_64-w64-mingw32 --enable-cxx --disable-shared --
↪enable-static --with-pic
make
```

3.2.4 Mac Pre-Installation Requirements (on MacOS Sierra)

Install dependencies on your operating system.

```
Install XCode and XCode command line tools
Install git from git-scm
Install brew (follow instructions on brew.sh)
brew install autoconf automake berkeley-db4 libtool boost@1.57 openssl pkg-config␣
↪rename
brew link boost@1.57 --force
```

Prepare for Static Linking

Apple does not support statically linked binaries as documented here. However, it is convenient for end-users to launch a binary without having to first install brew, a third-party system designed for developers.

To create a statically linked RecordsKeeper blockchain which only depends on default MacOS dylibs, the following steps are taken:

- Hide the brew boost dylibs from the build system: rename -e 's/.dylib/.dylib.hidden/' /usr/local/opt/boost/lib/*.dylib
- Hide the brew berekley-db dylibs from the build system: rename -e 's/.dylib/.dylib.hidden/' /usr/local/opt/berkeley-db@4/lib/*.dylib
- Hide the brew openssl dylibs from the build system: rename -e 's/.dylib/.dylib.hidden/' /usr/local/opt/openssl/lib/*.dylib

The default brew cookbook for berkeley-db and boost builds static libraries, but the default cookbook for openssl only builds dylibs.

- Tell brew to build openssl static libraries: brew edit openssl In 'def configure_args' change 'shared' to 'no-shared' brew install openssl –force

Building from Source

Clone the Repository

To clone the source code, execute the following command:

```
git clone https://github.com/RecordsKeeper/recordskeeper-core.git
cd recordskeeper-core
```

Compile RecordsKeeper for Mac (64-bit)

This will build rkd, rk-cli and rk-util in the src directory.

```
export LDFLAGS=-L/usr/local/opt/openssl/lib
export CPPFLAGS=-I/usr/local/opt/openssl/include
./autogen.sh
./configure --with-gui=no --with-libs=no --with-miniupnpc=no
make
```

Clean up

```
rename -e 's/.dylib.hidden/.dylib/' /usr/local/opt/berkeley-db\@4/lib/*.dylib.hidden
rename -e 's/.dylib.hidden/.dylib/' /usr/local/opt/boost/lib/*.dylib.hidden
rename -e 's/.dylib.hidden/.dylib/' /usr/local/opt/openssl/lib/*.dylib.hidden
brew edit openssl
    In 'def configure_args' change 'no-shared' to 'shared'
```

3.3 Getting Started

This tutorial requires you to setup a RecordsKeeper's public blockchain node on your local machine or remote server. If you have not done so already, please download and install RecordsKeeper's client on a local machine or remote server. If you are using the RecordsKeeper client on Windows, please read the installation instructions specific to Windows on the previous page.

Download the latest RecordsKeeper build and put the downlodeded files under the bin folder or any other folder and add its path to the System PATH environment variable so that you can run these commands from anywhere. Alternatively, you can go to the folder which contains RecordsKeeper binaries and execute these commands from there.

3.3.1 Connecting to the RecordsKeeper blockchain

To connect to the RecordsKeeper blockchain run the following command in bash or command prompt.

RecordsKeeper Testnet

```
rkd recordskeeper-test@35.170.155.89:8379 -daemon
```

RecordsKeeper Mainnet

```
rkd recordskeeper@35.172.1.247:7895 -daemon
```

When you run the above commands, they create a node connected to the RecordsKeeper Main node. The command will automatically set the permissions and consensus required for the blockchain.

RecordsKeeper Testnet is a complete open blockchain. When you connect to the Testnet, you will have the permissions to connect, send, receive and mine, but the RecordsKeeper Mainnet is currently under consensus rule for a certain number of blocks (You can read more about that in the RecordsKeeper Whitepaper) to avoid takeovers. When you connect to the RecordsKeeper Mainnet, you will receive an address. You can send that address to us through RecordsKeeper Mining Permission .

3.3.2 Interactive Command Line Mode

Before we proceed, let's enter interactive mode so we can issue commands to interact with the RecordsKeeper blockchain.

Testnet

```
rk-cli recordskeeper-test
```

Mainnet

```
rk-cli recordskeeper
```

3.3.3 Some Basic Commands

Now that your node is up and running you can issue some basic commands in the interactive mode to familiarize yourself with RecordsKeeper.

To access general information about the RecordsKeeper Node:

```
getinfo
```

See a list of all available commands for the RecordsKeeper Node:

```
help
```

Create a new address in the RecordsKeeper Node wallet:

```
getnewaddress
```

List all addresses in the RecordsKeeper Node wallet:

```
getaddresses
```

3.3.4 Sending a Transaction in RecordsKeeper

The RecordsKeeper blockchain works on the same backend as Bitcoin algorithms. Both the RecordsKeeper Testnet and Mainnet can be used to send and receive XRK tokens. Use the following interactive commands to send transactions on RecordsKeeper blockchain.

Send

```
send address amount (comment) (comment-to)
Example: send 1KJFg5YLpvYNYZtCM6hhNYW8uBKtc3GHVboXco 10
```

This command is used to send one or more XRK tokens to an address, returning the txid. The amount field is the quantity of XRK tokens, and the address field is the address where you want to send the XRK tokens. This command will use the node's root address to send the transaction. Please make sure you have sufficient balance in the Node's root address for transactions to propagate over the RecordsKeeper blockchain. You can also provide specific comments for the transaction which are optional. The fees will be applied as per the transaction size.

Send from a Different Address

```
sendfrom from-address to-address amount
Example: sendfrom 1KJFg5YLpvYNYZtCM6hhNYW8uBKtc3GHVboXco␣
↪17gddiicYtbnwnWuY2ZYvM1Rw9e7t3pPjNJPab 10
```

This command is also used to send one or more XRK tokens to address, returning the txid. Using this command you can specify the from-address which you want to use to send the transaction. The amount field is the quantity of XRK tokens and the to-address field is the address where you want to send the XRK tokens. Please make sure you have sufficient balance in the from-address for transaction to propagate over the RecordsKeeper blockchain. The from-address used here is also one of the addresses generated for the node. You can also provide specific comments for the transaction which are optional. The fees will be applied as per the transaction size.

3.3.5 Publishing and Retrieving Data in RecordsKeeper

The RecordsKeeper blockchain is a public key-value based database on the blockchain. You can use the interactive command line to publish and retrieve stored information. As the blockchain is a shared concept, you can view all the published data and retrieve it using only a key or address. RecordsKeeper uses the streams to store the data. RecordsKeeper streams provide a natural abstraction for the RecordsKeeper blockchain, which focuses on general data retrieval, timestamping, and archiving, rather than the transfer of tokens between participants.

The "root" stream is open to all and anyone can publish data into the root stream. The following commands will give you a brief introduction about how to work with data over RecordsKeeper blockchain.

Publish

```
publish stream key data-hex
Example: publish root rkKey 57687920796f7520636f6e766572746564206d653f
```

This command publishes an item in a stream. The stream name is passed, with the key provided in text form and a data-hex in hexadecimal format. It returns ref or creation txid. The data is published using the node's address. Use the next command discussed below to publish data from different address. The mining fees are applied as per the transaction size.

Publish from a Different Address

```
publishfrom from-address stream key data-hex
Example: publishfrom 1KJFg5YLpvYNYZtCM6hhNYW8uBKtc3GHVboXco root rkKey␣
↪596f7520636f6e766572746564206d6520616761696e3f
```

This command works like publish, but publishes the item from the from-address. It is useful if the node has multiple addresses with different amounts. The mining fees are applied as per the transaction size.

Send as a Transaction

```
sendwithdata/sendwithmetadata address amount data-hex|object
Example: sendwithdata 1KJFg5YLpvYNYZtCM6hhNYW8uBKtc3GHVboXco 0 {"for":root,"key":
↪"rkKey","data":"506c656173652073746f7020646f696e67207468697321"}
```

This works similarly to send, but with an additional data-only transaction output. You can pass raw data as data-hex hexadecimal string. It is also used to publish the data to a stream, pass an object like this {"for":StreamName,"key":"KeyName","data":"DataHex"} where stream is a stream name, ref or creation txid, the key is in text form, and the data is hexadecimal. You can pass the amount as 0, if you are only using this to publish the data over the RecordsKeeper stream. You can also send some XRK tokens while publishing the data over the stream. The fees will be applied as per the transaction size.

3.4 Mining Guide for Recordskeeper Blockchain

The following guide explains the process to start mining in different operating systems. Feel free to contact RecordsKeeper team for any queries related to same at:

- Facebook
- Twitter
- Telegram

We are always looking to improve and provide best solution for Blockchain.

3.4.1 Mining Guide for RecordsKeeper Blockchain on Linux

The following document helps the users to initiate mining for RecordsKeeper blockchain on the Linux operating system. All the commands and processes displayed in this document have been tested and created on the Ubuntu Operating System. The detailed overview to start mining for RecordsKeeper blockchain is as follows:

System Requirements

- Linux: 64-bit, supports Ubuntu 12.04+, CentOS 6.2+, Debian 7+, Fedora 15+, RHEL 6.2+.
- 512 MB of RAM
- 1 GB of disk space

Installing RecordsKeeper on Linux

First, install these dependencies by executing the following commands:

```
sudo apt-get update
sudo apt-get install build-essential libtool autotools-dev automake pkg-config libssl-
↪dev libevent-dev bsdmainutils
sudo apt-get install libboost-all-dev
sudo add-apt-repository ppa:bitcoin/bitcoin
sudo apt-get update
sudo apt-get install libdb4.8-dev libdb4.8++-dev
```

To download the executable directly from the browser click here .

And, if you want to download it from the command line terminal, then use this command:

```
wget https://github.com/RecordsKeeper/recordskeeper-core/releases/download/v1.0.0/
↪recordskeeper-1.0.0.tar.gz
```

Then, move to the location of the downloaded files and run the following commands from your terminal:

```
tar -xvzf recordskeeper-1.0.0.tar.gz
cd recordskeeper-1.0.0
mv rkd rk-cli rk-util /usr/local/bin
```

Moving the RecordsKeeper files to the bin directory to make them easily accessible from the command line anywhere.

Note:

- If you get an error, then run the above commands using "sudo" for root privileges
- Use exit command (to return to your regular user)
- Linux users move directly to the connecting-rk section

Connecting to RecordsKeeper Blockchain on Linux

The RecordsKeeper Testnet blockchain is available for users to develop and deploy applications on. XRK Testnet tokens do not hold any value and are only available for testing. You can earn XRK tokens from RecordsKeeper Mainnet mining.

Now, to connect to the RecordsKeeper blockchain, run the following command from the terminal:

RecordsKeeper Testnet

```
rkd recordskeeper-test@35.170.155.89:8379
```

RecordsKeeper Mainnet

```
rkd recordskeeper@35.172.1.247:7895
```

This command will initialize your node.

And, if you want your connection to remain active as a background process then run this command:

RecordsKeeper Testnet

```
rkd recordskeeper-test@35.172.1.247:8379 -daemon
```

RecordsKeeper Mainnet

```
rkd recordskeeper@35.172.1.247:7895 -daemon
```

Note: Linux users can now go to the mining-permissions section.

Mining Permissions

Running RecordsKeeper on Linux

You will see the following message on your Linux command line terminal after you execute the command to connect to the RecordsKeeper blockchain.

RecordsKeeper Permissions

RecordsKeeper Testnet

The mining for RecordsKeeper Testnet is open for everyone, so when you connect to RecordsKeeper Testnet, you will receive all the permissions for your default address

RecordsKeeper Mainnet

For Mainnet, when your node gets connected, you will receive the permissions to connect, send, and receive. Now look for your default XRK address from the command given below, which will display your node's wallet address. This address is your "default XRK address" or "public address" on the RecordsKeeper blockchain in which you will receive XRK tokens. To check the address, run the following command:

```
rk-cli recordskeeper getaddresses
```

Submit the following to receive mining permissions for RecordsKeeper Mainnet.

Note: Copy the above generated address and send it to us here .

Only after the RecordsKeeper team grant mining permissions to your node address will you be able to mine XRK tokens into your default address.

To retreive the private key for your node address, run this command:

```
rk-cli recordskeeper dumpprivkey {default_XRK_address}        #(input node_address␣
↪without braces)
```

Note: Please store this private key safely. Losing it will result in the loss of XRK tokens.

After completing the above process, you can check for your node's information (best block and synced block) by running the following commands:

```
rk-cli recordskeeper getinfo                  #(for synced block)
rk-cli recordskeeper getblockchaininfo        #(for best block)
```

Your node will sync with the best block, and then only your node can start mining and your balance will get updated with the mined XRK tokens.

If you have entered the wrong IP-address, then it will report this error:

Warning: Error: Couldn't initialize permission database for blockchain recordskeeper. Probably rkd for this blockchain is already running. Exiting…

Please check the IP address and port properly to connect to the RecordsKeeper blockchain.

Note: If you have already created a wallet address and you want to add it as your miner address then run this command from the command line terminal:

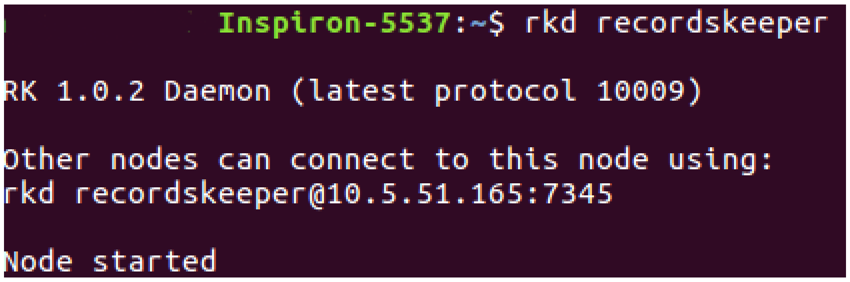
```
rk-cli recordskeeper importprivkey {private_key}       #(include private key without␣
↪braces)
```

Connecting to RecordsKeeper Blockchain after Permissions

Once the permissions for the RecordsKeeper Mainnet have been granted, you can directly connect to the RecordsKeeper chain and see your mining progress. You can run the following commands to connect to the RecordsKeeper blockchain and view the mining address.

As the IP configuration was already stored with you when you intiated the connection, you can directly run the following command:

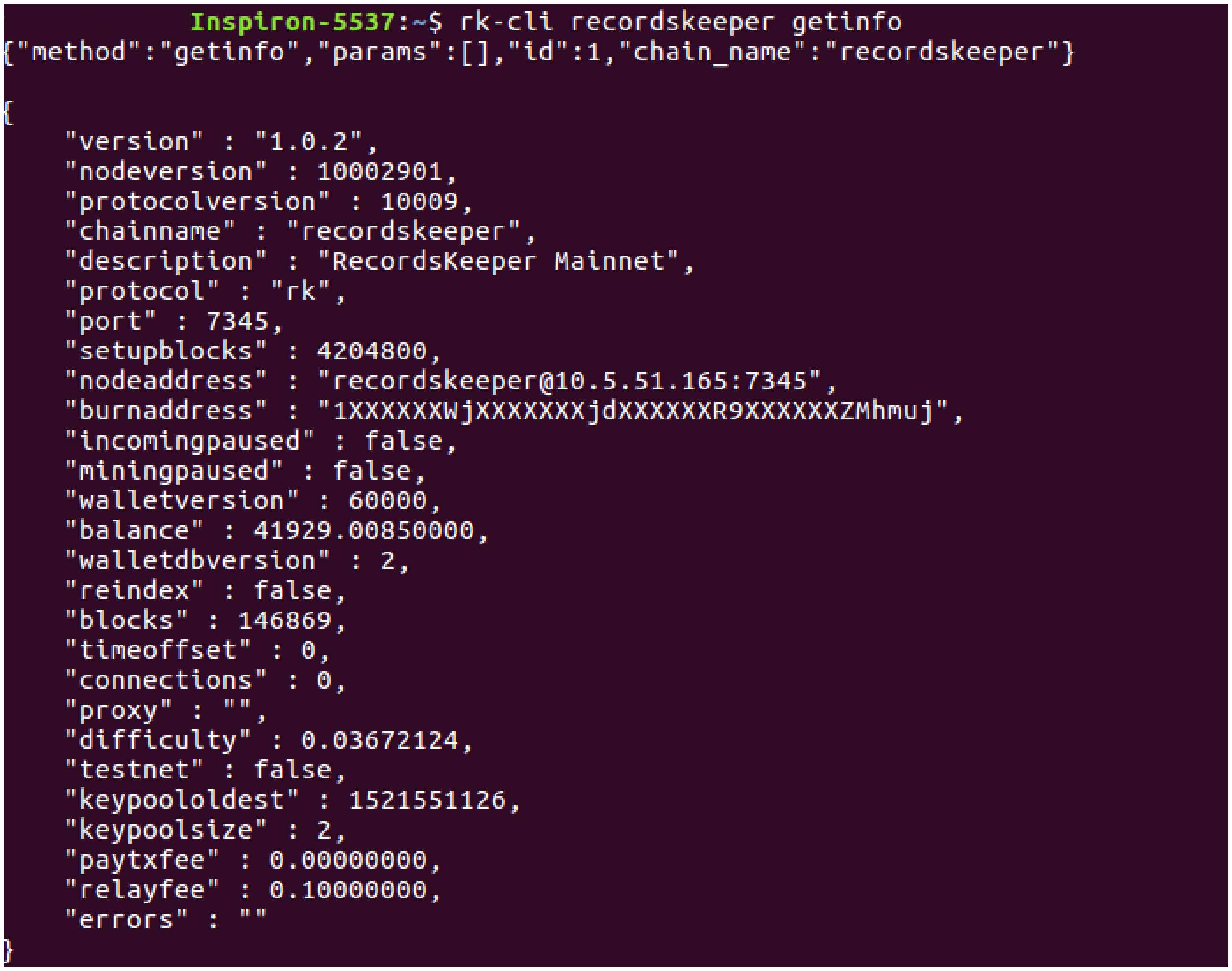
```
rkd recordskeeper -daemon
```



```
Inspiron-5537:~$ rkd recordskeeper

RK 1.0.2 Daemon (latest protocol 10009)

Other nodes can connect to this node using:
rkd recordskeeper@10.5.51.165:7345

Node started
```

You can run getinfo command or getaddressbalances command to see the balance in your node or directly the node address.

```
rk-cli recordskeeper getinfo
```

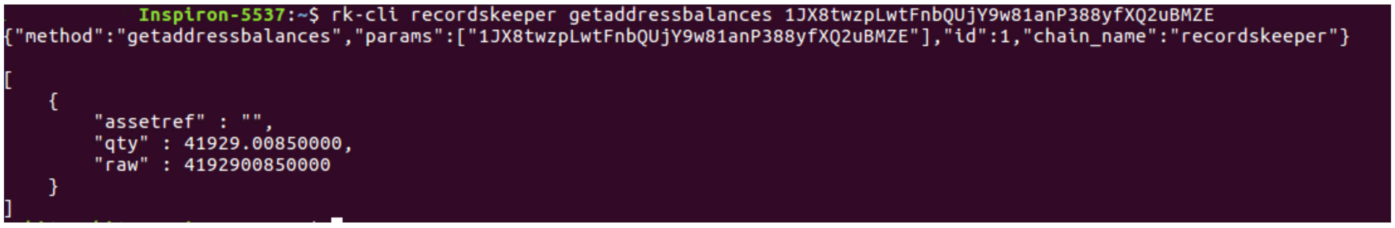


```
            Inspiron-5537:~$ rk-cli recordskeeper getinfo
{"method":"getinfo","params":[],"id":1,"chain_name":"recordskeeper"}

{
    "version" : "1.0.2",
    "nodeversion" : 10002901,
    "protocolversion" : 10009,
    "chainname" : "recordskeeper",
    "description" : "RecordsKeeper Mainnet",
    "protocol" : "rk",
    "port" : 7345,
    "setupblocks" : 4204800,
    "nodeaddress" : "recordskeeper@10.5.51.165:7345",
    "burnaddress" : "1XXXXXXWjXXXXXXXjdXXXXXXR9XXXXXXZMhmuj",
    "incomingpaused" : false,
    "miningpaused" : false,
    "walletversion" : 60000,
    "balance" : 41929.00850000,
    "walletdbversion" : 2,
    "reindex" : false,
    "blocks" : 146869,
    "timeoffset" : 0,
    "connections" : 0,
    "proxy" : "",
    "difficulty" : 0.03672124,
    "testnet" : false,
    "keypoololdest" : 1521551126,
    "keypoolsize" : 2,
    "paytxfee" : 0.00000000,
    "relayfee" : 0.10000000,
    "errors" : ""
}
```

Note: You can view your balances in the balance output of the getinfo command.

OR

```
rk-cli recordskeeper getaddressbalances <Your Node Address Given for Mining>
```



```
            Inspiron-5537:~$ rk-cli recordskeeper getaddressbalances 1JX8twzpLwtFnbQUjY9w81anP388yfXQ2uBMZE
{"method":"getaddressbalances","params":["1JX8twzpLwtFnbQUjY9w81anP388yfXQ2uBMZE"],"id":1,"chain_name":"recordskeeper"}

[
    {
        "assetref" : "",
        "qty" : 41929.00850000,
        "raw" : 4192900850000
    }
]
```

Note: Please do not use the address specified above. This address is only available for the demo purpose.

Stopping RecordsKeeper Blockchain

RecordsKeeper Mainnet

In case you want to stop your running RecordsKeeper node, you can use the following command from your command line terminal:

```
rk-cli recordskeeper stop
```

RecordsKeeper Testnet

In case you want to stop your RecordsKeeper-test blockchain node, you can use the following command from your command line terminal:

```
rk-cli recordskeeper-test stop
```

3.4.2 Mining Guide for RecordsKeeper Blockchain on Windows

The following document helps the users to initiate mining the RecordsKeeper blockchain using a Windows Operating system. All the commands and processes displayed in this document have been tested and created using Windows 7 and above operating systems. A detailed overview to start mining the RecordsKeeper blockchain is as follows:

- *System Requirements*
- *Installing RecordsKeeper on Windows*
- *Connecting to RecordsKeeper Blockchain on Windows*
- *Mining Permissions*
- *Connecting to RecordsKeeper Blockchain after Permissions*
- *Stopping RecordsKeeper Blockchain*

System Requirements

- Windows: 64-bit, supports Windows 7, 8, 10, Server 2008 or later.
- 512 MB of RAM
- 1 GB of disk space

Installing RecordsKeeper on Windows

Download the executables from here and then unzip the folder and you will have the binary files: rkd.exe, rkd-cold.exe, rk-cli.exe and rk-util.exe.

Copy the folder to your desired location.

Then, open your command line terminal and go to that location. After that run the following command:

```
cd recordskeeper-windows-1.0.0
```

Note: Windows users move directly to the connecting-rk section.

Connecting to RecordsKeeper Blockchain on Windows

The RecordsKeeper Testnet blockchain is avaialble for users to develop and deploy applications on the RecordsKeeper blockchain. XRK Testnet tokens do not hold any value and are only available for testing. You can earn XRK tokens from RecordsKeeper Mainnet mining.

Now, to connect to the RecordsKeeper blockchain, first go into the directory where you have downloaded "recordskeeper-windows-1.0.0.zip" and then open the command line terminal from that location:

RecordsKeeper Testnet

```
rkd recordskeeper-test@35.170.155.89:8379
```

RecordsKeeper Mainnet

```
rkd recordskeeper@35.172.1.247:7895
```

This command will initialize your node.

And, if you want your connection to remain active as a background process, then run this command:

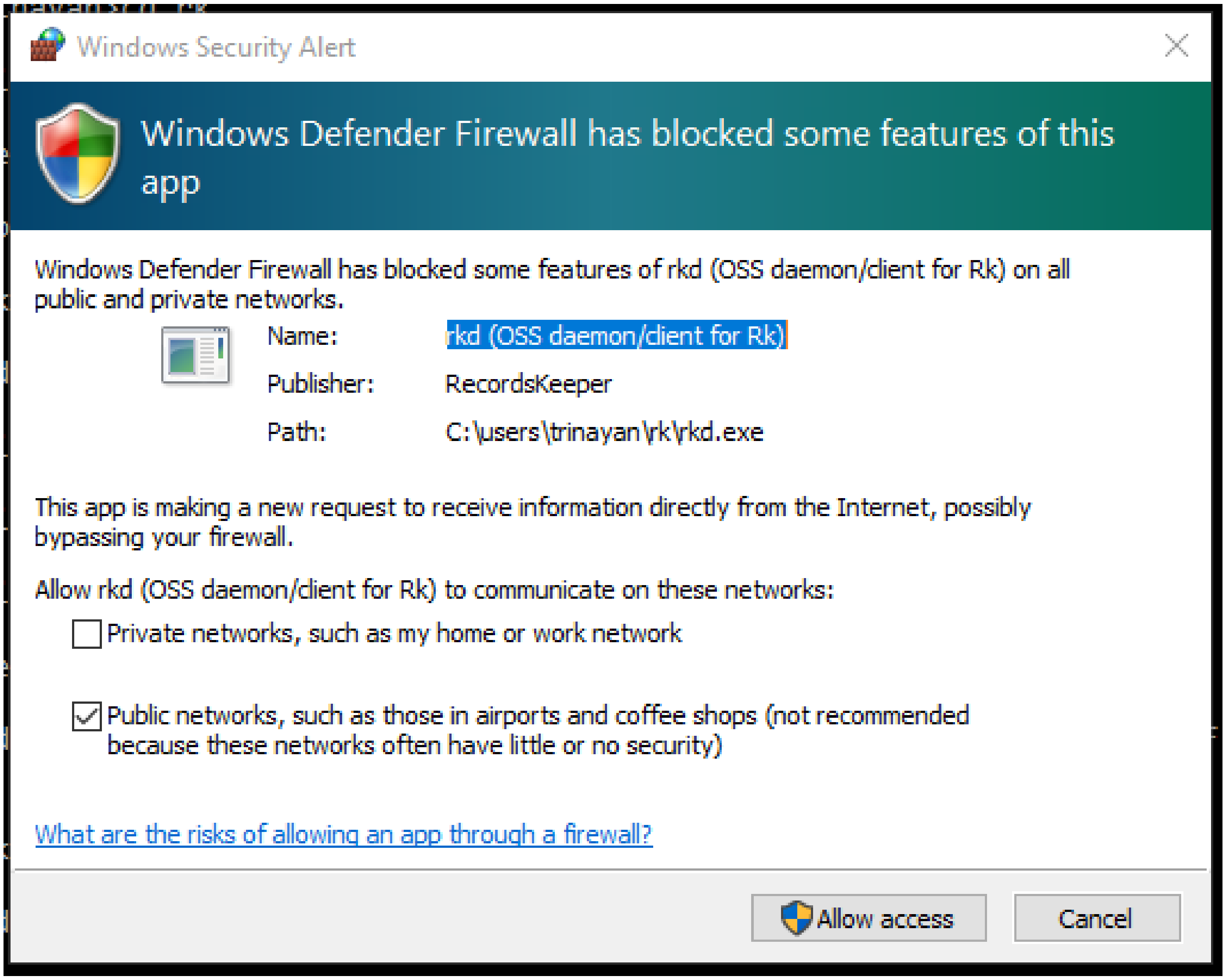
RecordsKeeper Testnet

```
rkd recordskeeper-test@35.170.155.89:8379 -daemon
```

RecordsKeeper Mainnet

```
rkd recordskeeper@35.172.1.247:7895 -daemon
```

If the operating system does not pop up the allow firewall connections for rkd like shown below:


Windows Security Alert

Windows Defender Firewall has blocked some features of this app

Windows Defender Firewall has blocked some features of rkd (OSS daemon/client for Rk) on all public and private networks.

Name: rkd (OSS daemon/client for Rk)

Publisher: RecordsKeeper

Path: C:\users\trinayan\rk\rkd.exe

This app is making a new request to receive information directly from the Internet, possibly bypassing your firewall.

Allow rkd (OSS daemon/client for Rk) to communicate on these networks:

☐ Private networks, such as my home or work network

☑ Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)

What are the risks of allowing an app through a firewall?

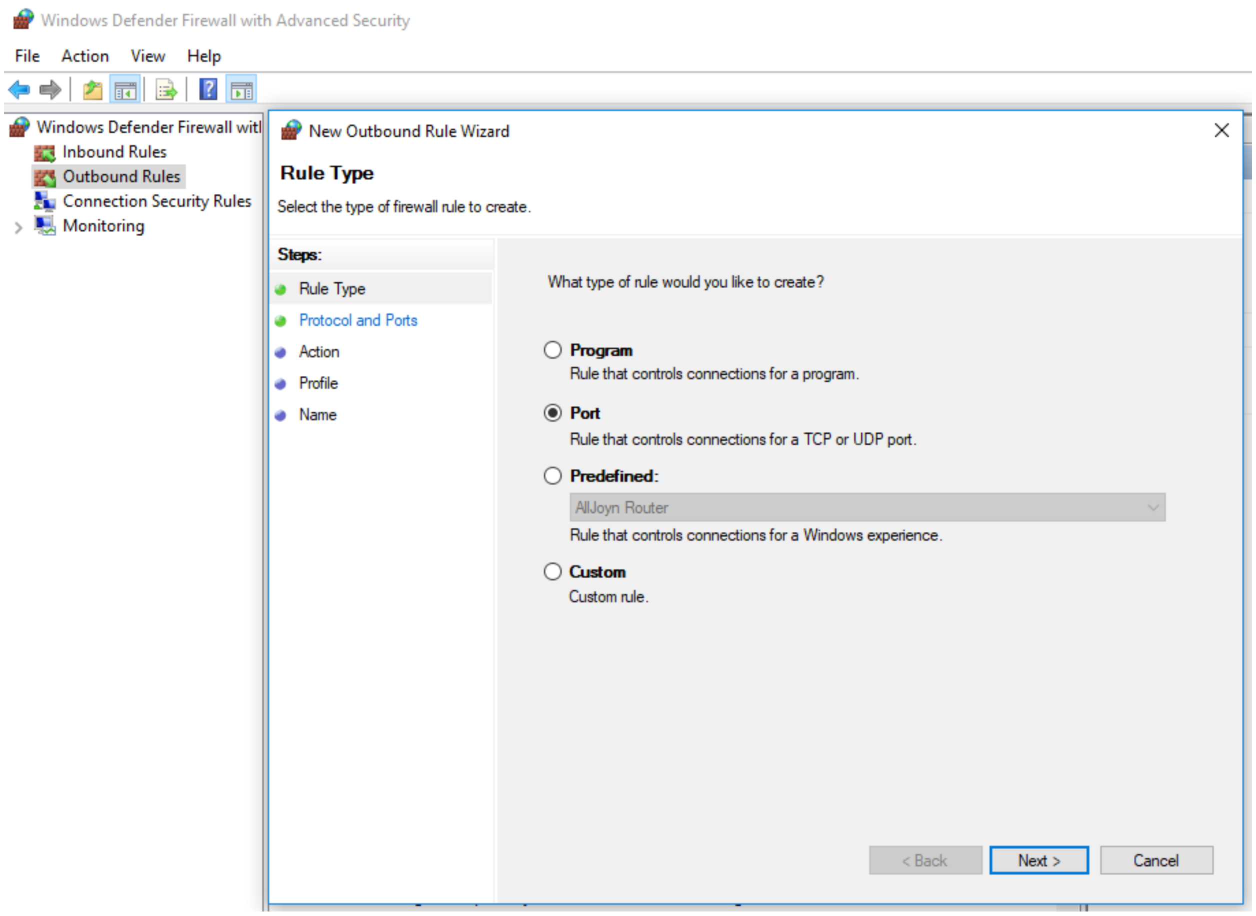
Then, you have to manually allow connections through your firewall by following the steps below:

Go to:

```
Control Panel > System and Security > Windows Defender Firewall > Advanced Settings
```

And add Outbound Rules by following these steps:

Step 1: First select the rule type. As you have to create a rule for allowing the network port, select Port here and then press Next:



Rule Type

Select the type of firewall rule to create.

What type of rule would you like to create?

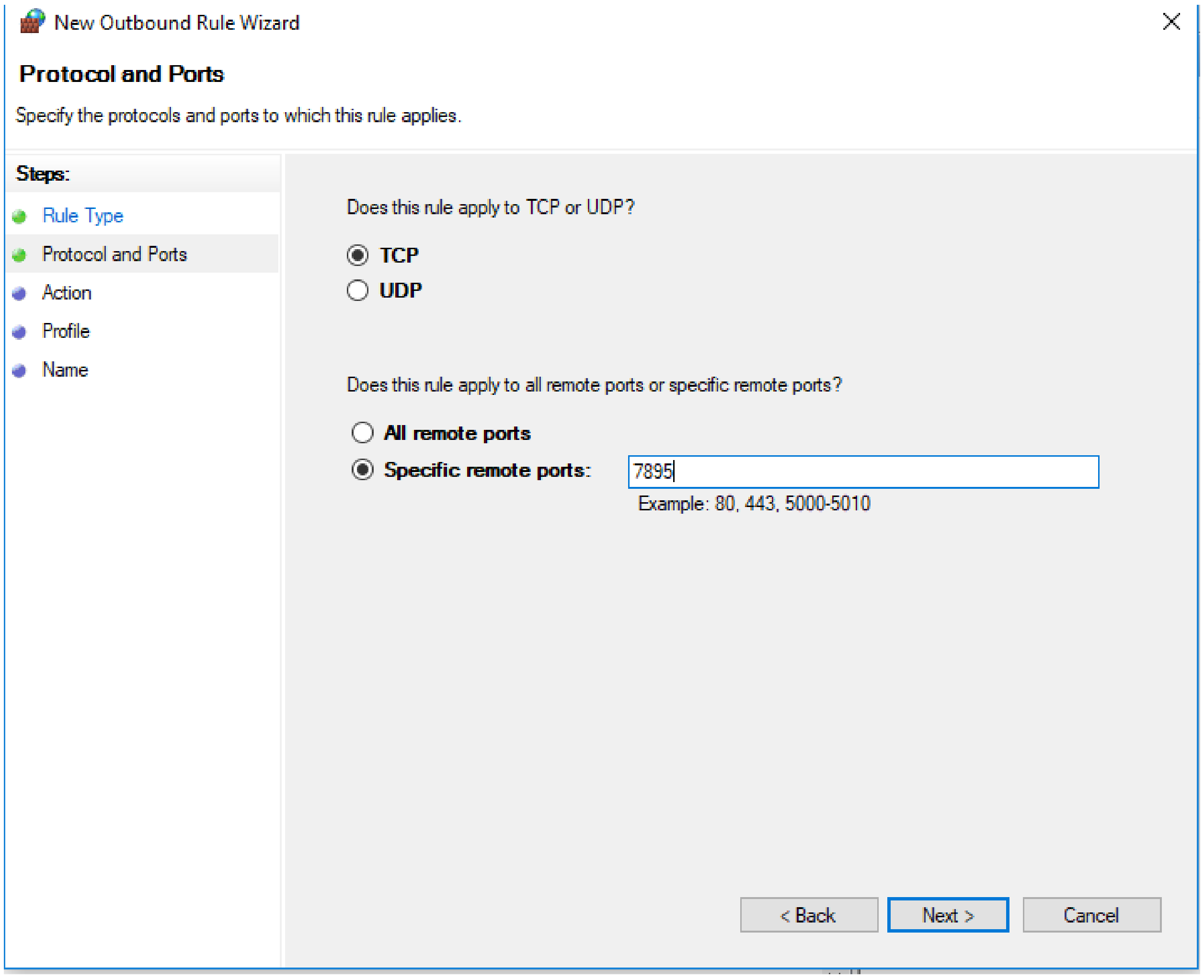
○ **Program**
Rule that controls connections for a program.

● **Port**
Rule that controls connections for a TCP or UDP port.

○ **Predefined:**
AllJoyn Router
Rule that controls connections for a Windows experience.

○ **Custom**
Custom rule.

Step 2: Specify the port address of the RecordsKeeper blockchain to which you are allowing access, for Testnet type 8379 and to open ports for Mainnet type 7895 in the textbox, and then click on Next:


New Outbound Rule Wizard

Protocol and Ports

Specify the protocols and ports to which this rule applies.

Steps:
- Rule Type
- Protocol and Ports
- Action
- Profile
- Name

Does this rule apply to TCP or UDP?

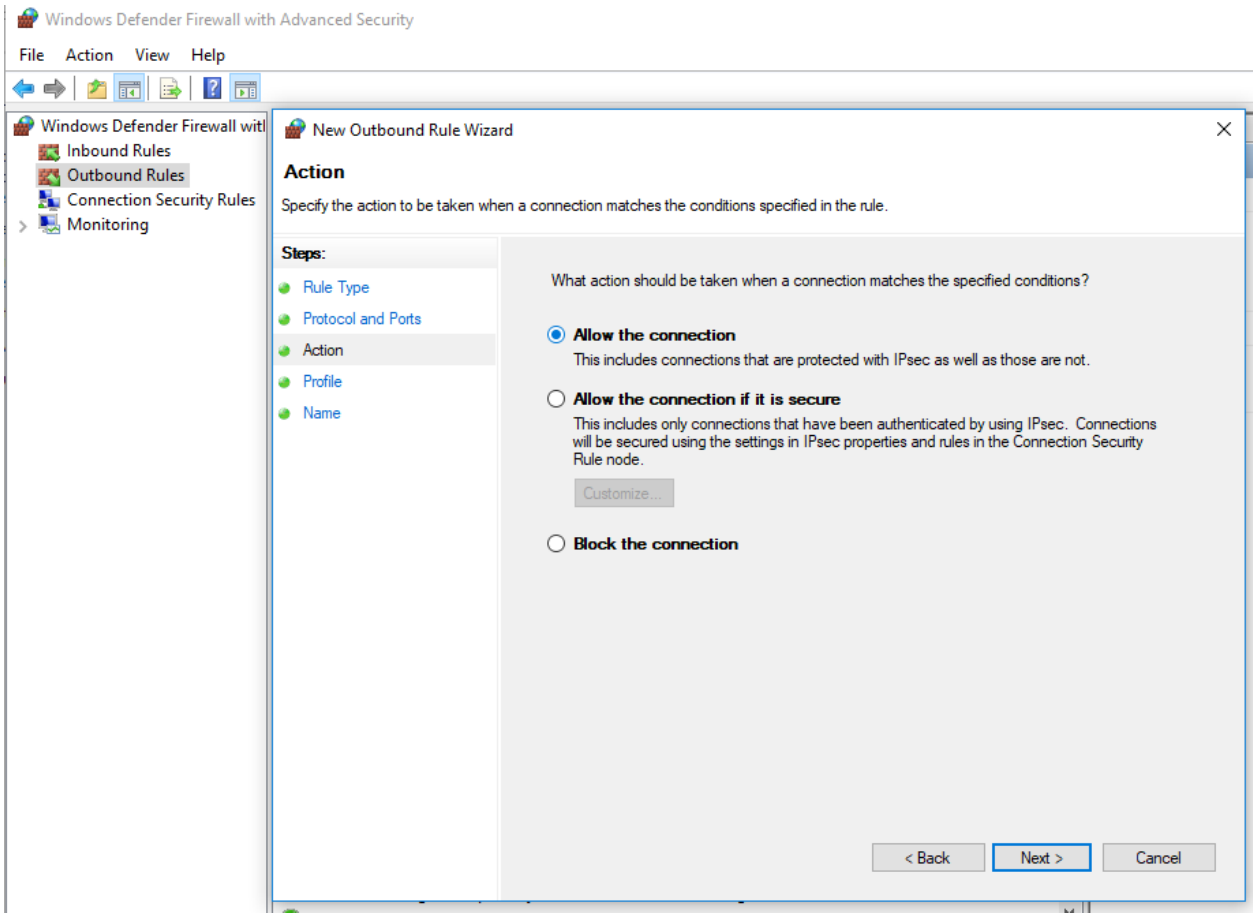
[✓ TCP] UDP

Does this rule apply to all remote ports or specific remote ports?

All remote ports [✓ Specific remote ports:] 7895

Example: 80, 443, 5000-5010

Step 3: Now click on *Allow the connection* and then press Next:



Action

Specify the action to be taken when a connection matches the conditions specified in the rule.

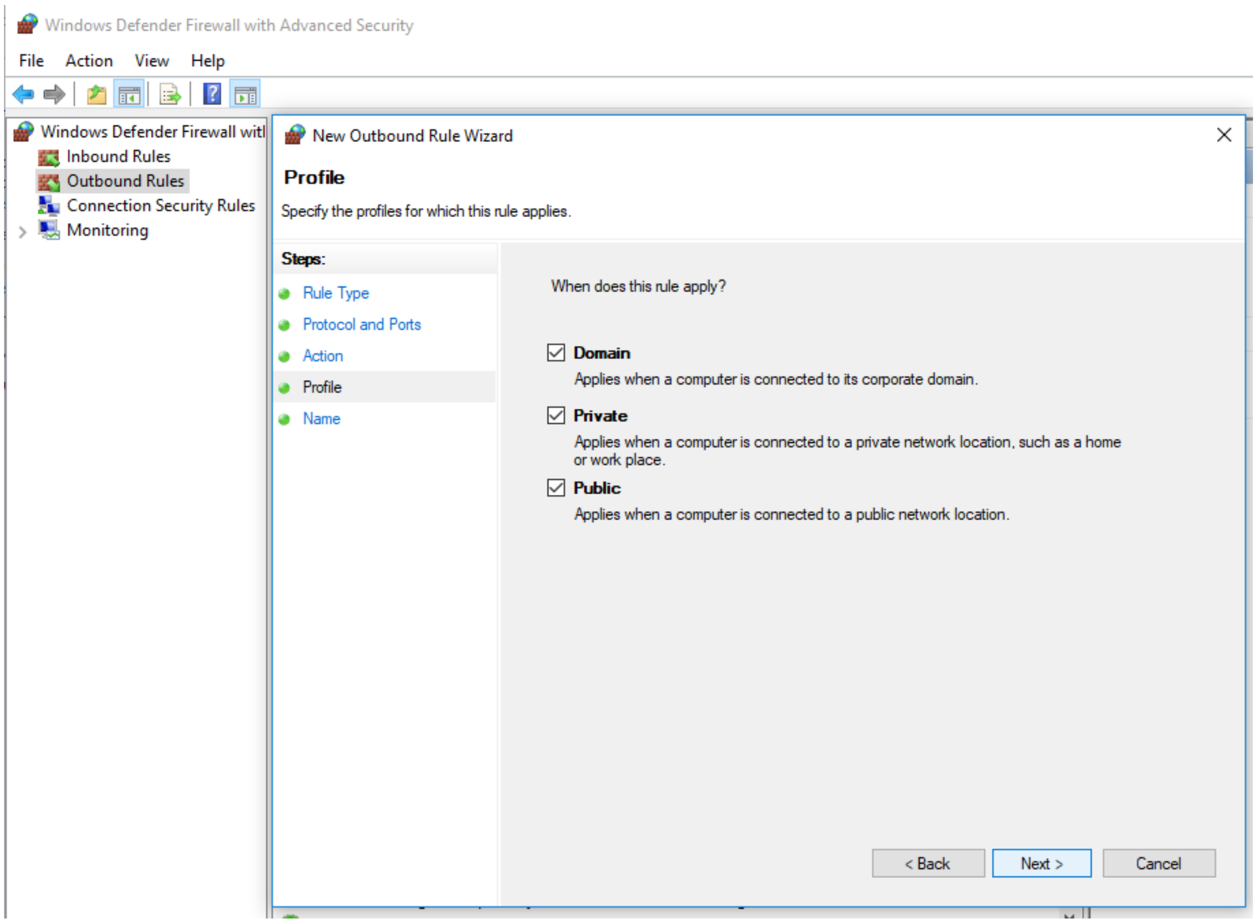
What action should be taken when a connection matches the specified conditions?

(●) **Allow the connection**
This includes connections that are protected with IPsec as well as those are not.

( ) **Allow the connection if it is secure**
This includes only connections that have been authenticated by using IPsec. Connections will be secured using the settings in IPsec properties and rules in the Connection Security Rule node.

( ) **Block the connection**

Step 4: Select all three profiles here for the rule to apply, and then click on Next:



Profile

Specify the profiles for which this rule applies.

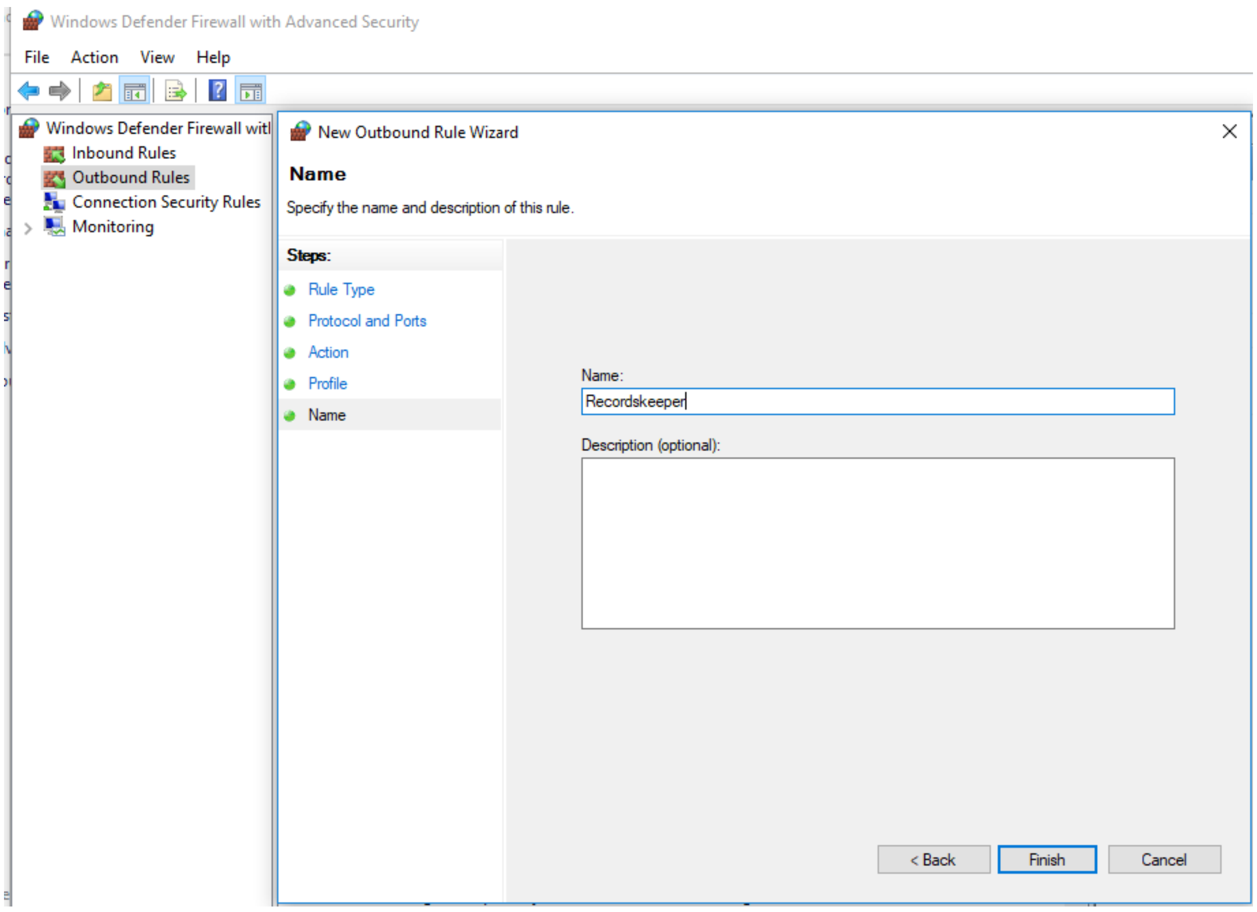
When does this rule apply?

☑ **Domain**
Applies when a computer is connected to its corporate domain.

☑ **Private**
Applies when a computer is connected to a private network location, such as a home or work place.

☑ **Public**
Applies when a computer is connected to a public network location.

Step 5: Now choose a name for the created Outbound rule and then press *Finish* to complete the process of opening up the ports.



Name

Specify the name and description of this rule.

Name: Recordskeeper

Description (optional):

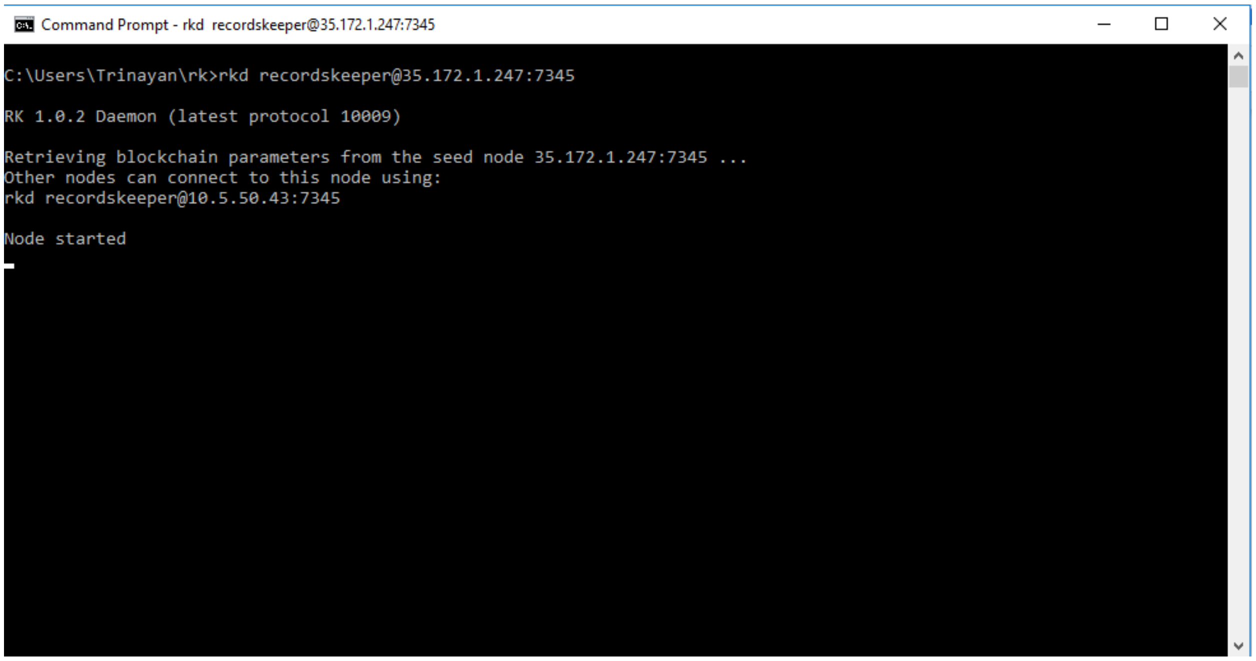
Note: If you want another RecordsKeeper node to connect to your node then you have to allow connections by making Inbound rules for the same. Follow the same procedure after selecting a new Inbound rule.

Note: Windows users now can go to the mining-permissions section.

Mining Permissions

Running RecordsKeeper on Windows

You will see the following message on your Windows command line terminal after you execute the command to connect to the RecordsKeeper blockchain.



```
C:\Users\Trinayan\rk>rkd recordskeeper@35.172.1.247:7345

RK 1.0.2 Daemon (latest protocol 10009)

Retrieving blockchain parameters from the seed node 35.172.1.247:7345 ...
Other nodes can connect to this node using:
rkd recordskeeper@10.5.50.43:7345

Node started
```

Note: Windows users have to open a new command line terminal window for running the remaining commands.

RecordsKeeper Permissions

RecordsKeeper Testnet

The mining for RecordsKeeper Testnet is open to everyone, so when you connect to RecordsKeeper Testnet, you will receive all the permissions for your default address

RecordsKeeper Mainnet

For Mainnet, when your node gets connected, you will receive the permissions to connect, send, and receive. Now look for your default XRK address from the command given below, which will display your node's wallet address. This address is your "default XRK address" or "public address" of the RecordsKeeper blockchain in which you will receive XRK tokens. To check the address, run the following command:

```
rk-cli recordskeeper getaddresses
```

Submit the following to receive Mining Permissions for RecordsKeeper Mainnet.

Note: Copy the above generated address and send it to us here .

Only after the RecordsKeeper team grants mining permissions to your node address will you be able to mine XRK tokens into your default address.

To retrieve the private key for your node address, run this command:

```
rk-cli recordskeeper dumpprivkey {default_XRK_address}        #(input node_address␣
↪without braces)
```

Note: Please store this private key safely. Losing it will result in the loss of XRK tokens.

After completing the above process, you can check for your node's information (best block and synced block) by running the following commands:

```
rk-cli recordskeeper getinfo                  #(for synced block)
rk-cli recordskeeper getblockchaininfo        #(for best block)
```

Your node will sync with the best block, and then only your node can start mining and your balance will get updated with the mined XRK tokens.

If you have entered the wrong IP address, it will report this error:

Warning: Error: Couldn't initialize permission database for blockchain recordskeeper. Probably rkd for this blockchain is already running. Exiting…

Please check the IP address and port properly to connect to the RecordsKeeper blockchain.

Note: If you have already created a wallet address and you want to add it as your miner address, then run this command from the command line terminal:

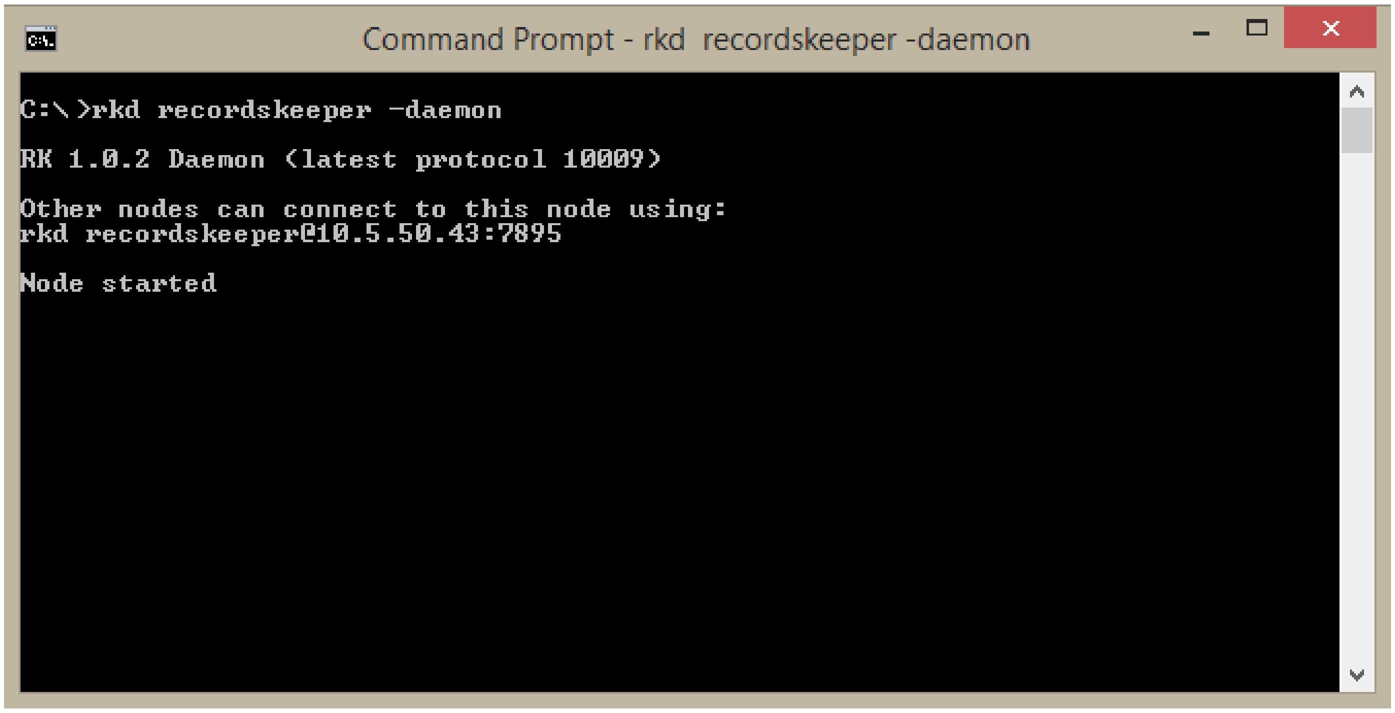
```
rk-cli recordskeeper importprivkey {private_key}      #(include private key without␣
↪braces)
```

Connecting to RecordsKeeper Blockchain after Permissions

Once the permissions for RecordsKeeper Mainnet have been granted, you can directly connect to the RecordsKeeper chain and see your mining progress. You can run the following commands to connect to the RecordsKeeper blockchain and view the mining address.

As the IP configuration was already stored with you when you initiated the connection, you can directly run the following command:

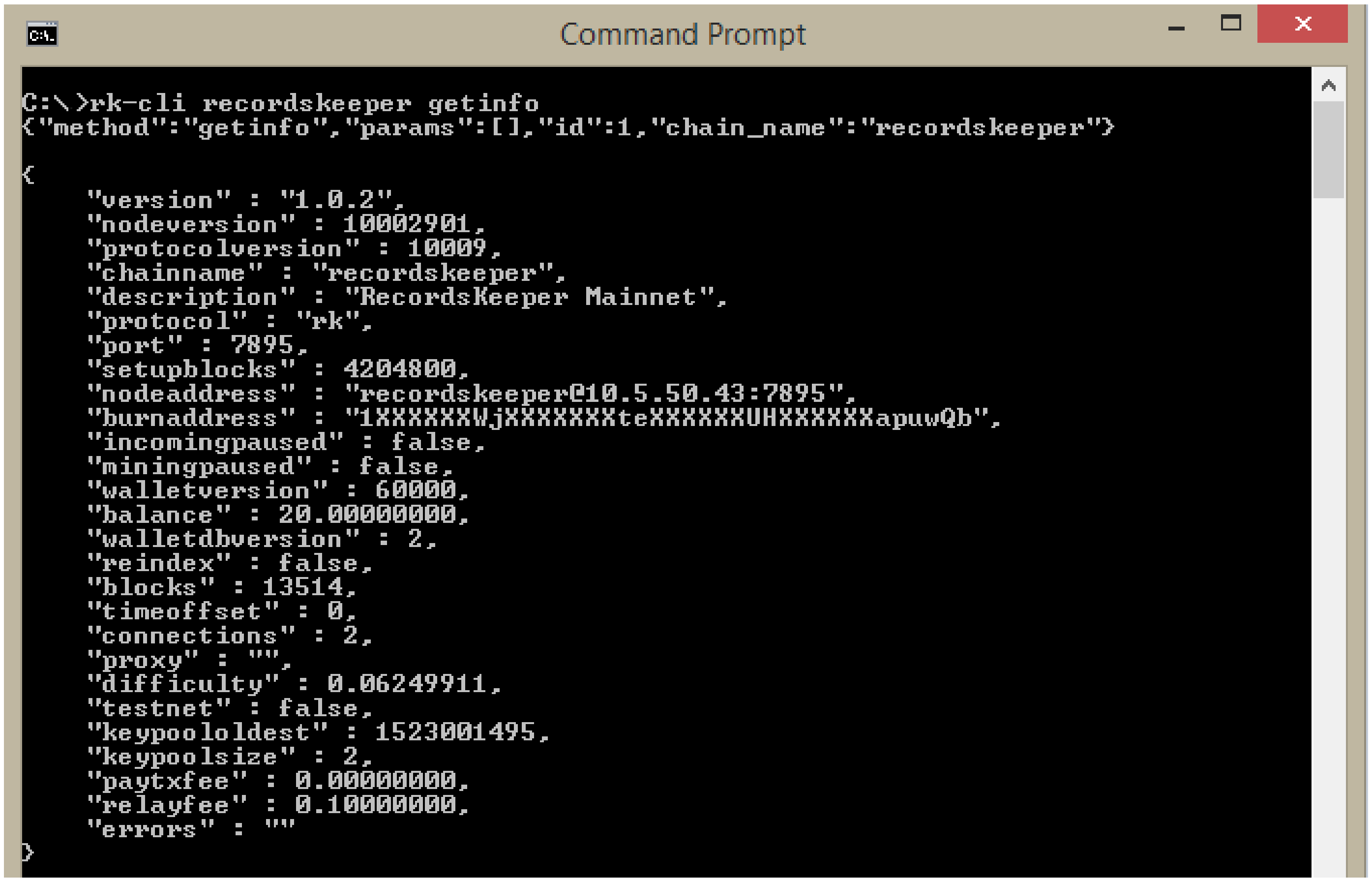
```
rkd recordskeeper -daemon
```


Command Prompt - rkd recordskeeper -daemon

```
C:\>rkd recordskeeper -daemon

RK 1.0.2 Daemon (latest protocol 10009)

Other nodes can connect to this node using:
rkd recordskeeper@10.5.50.43:7895

Node started
```

You can run the getinfo command or getaddressbalances command to see the balance in your node or the node address.

```
rk-cli recordskeeper getinfo
```

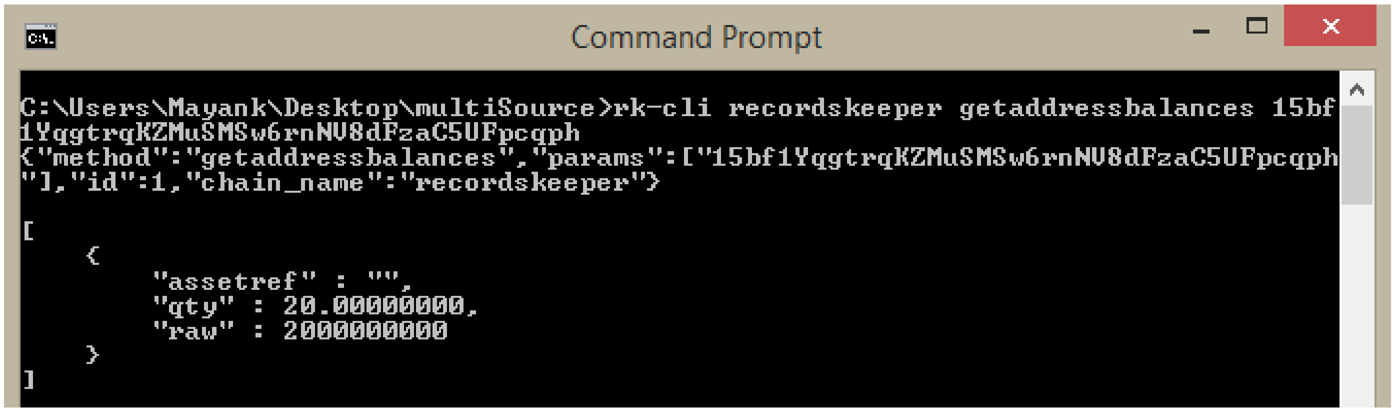

Command Prompt

```
C:\>rk-cli recordskeeper getinfo
{"method":"getinfo","params":[],"id":1,"chain_name":"recordskeeper"}

{
    "version" : "1.0.2",
    "nodeversion" : 10002901,
    "protocolversion" : 10009,
    "chainname" : "recordskeeper",
    "description" : "RecordsKeeper Mainnet",
    "protocol" : "rk",
    "port" : 7895,
    "setupblocks" : 4204800,
    "nodeaddress" : "recordskeeper@10.5.50.43:7895",
    "burnaddress" : "1XXXXXXWjXXXXXXXteXXXXXXUHXXXXXXapuwQb",
    "incomingpaused" : false,
    "miningpaused" : false,
    "walletversion" : 60000,
    "balance" : 20.00000000,
    "walletdbversion" : 2,
    "reindex" : false,
    "blocks" : 13514,
    "timeoffset" : 0,
    "connections" : 2,
    "proxy" : "",
    "difficulty" : 0.06249911,
    "testnet" : false,
    "keypoololdest" : 1523001495,
    "keypoolsize" : 2,
    "paytxfee" : 0.00000000,
    "relayfee" : 0.10000000,
    "errors" : ""
}
```

Note: You can view your balances in the balance output of the getinfo command.

OR

```
rk-cli recordskeeper getaddressbalances <Your Node Address Given for Mining>
```



```
C:\Users\Mayank\Desktop\multiSource>rk-cli recordskeeper getaddressbalances 15bf
1YqgtrqKZMuSMSw6rnNV8dFzaC5UFpcqph
{"method":"getaddressbalances","params":["15bf1YqgtrqKZMuSMSw6rnNV8dFzaC5UFpcqph
"],"id":1,"chain_name":"recordskeeper"}

[
    {
        "assetref" : "",
        "qty" : 20.00000000,
        "raw" : 2000000000
    }
]
```

Note: Please do not use the address specified above. This address is only available for the demo purpose.

Stopping RecordsKeeper Blockchain

RecordsKeeper Mainnet

If you want to stop your RecordsKeeper node, you can use the following command from your command line terminal:

```
rk-cli recordskeeper stop
```

RecordsKeeper Testnet

Ifyou want to stop your RecordsKeeper-test blockchain node, you can use the following command from your command line terminal:

```
rk-cli recordskeeper-test stop
```

3.4.3 Mining Guide for RecordsKeeper Blockchain on Mac

The following document helps the users to initiate mining the RecordsKeeper blockchain on a Mac operating system. All the commands and processes displayed in this document have been tested and created on Mac OS X 10.12 and above. The detailed overview to start mining for RecordsKeeper blockchain is as follows:

- *System Requirements*
- *Installing RecordsKeeper on Mac OS:*
- *Connecting to RecordsKeeper Blockchain on Mac*
- *Mining Permissions*
- *Connecting to RecordsKeeper Blockchain after Permissions*

- *Stopping Blockchain*

System Requirements

- Mac: 64-bit, supports OS X 10.12 (we hope to support earlier versions soon).
- 512 MB of RAM
- 1 GB of disk space

Installing RecordsKeeper on Mac OS:

First, install these dependencies by executing the following commands:

```
Install XCode and XCode command line tools
Install git from git-scm
Install brew (follow instructions on brew.sh)
brew install autoconf automake berkeley-db4 libtool boost@1.57 openssl pkg-config␣
↪rename
brew link boost@1.57 --force
```

To download the executable directly from the browser click here .

Unzip the zip file and then move to the location of the downloaded files and run the following commands from your terminal:

```
cd recordskeeper-mac-osx-1.0.0
mv rkd rk-cli rk-util /usr/local/bin
```

Moving the RecordsKeeper files to the bin directory makes them easily accessible from the command line anywhere.

Note:

- If you get an error, then run the above commands using "sudo" for root privileges
- Use exit command (to return to your regular user)
- Mac users move directly to the connecting-rk section

Connecting to RecordsKeeper Blockchain on Mac

The RecordsKeeper Testnet blockchain is available for users to develop and deploy applications on the RecordsKeeper blockchain. XRK Testnet tokens do not hold any value and are only available for testing. You can earn XRK tokens from RecordsKeeper Mainnet mining.

Now, to connect to the RecordsKeeper blockchain, run the following command from the terminal:

RecordsKeeper Testnet

```
./rkd recordskeeper-test@35.170.155.89:8379
```

RecordsKeeper Mainnet

```
./rkd recordskeeper@35.172.1.247:7895
```

This command will initialize your node.

And, if you want your connection to remain active as a background process, then run this command:

RecordsKeeper Testnet

```
./rkd recordskeeper-test@35.172.1.247:8379 -daemon
```

RecordsKeeper Mainnet

```
./rkd recordskeeper@35.172.1.247:7895 -daemon
```

In case of an error message like this:

Warning: Error: Couldn't initialize permission database for blockchain recordskeeper. Probably rkd for this blockchain is already running. Exiting…

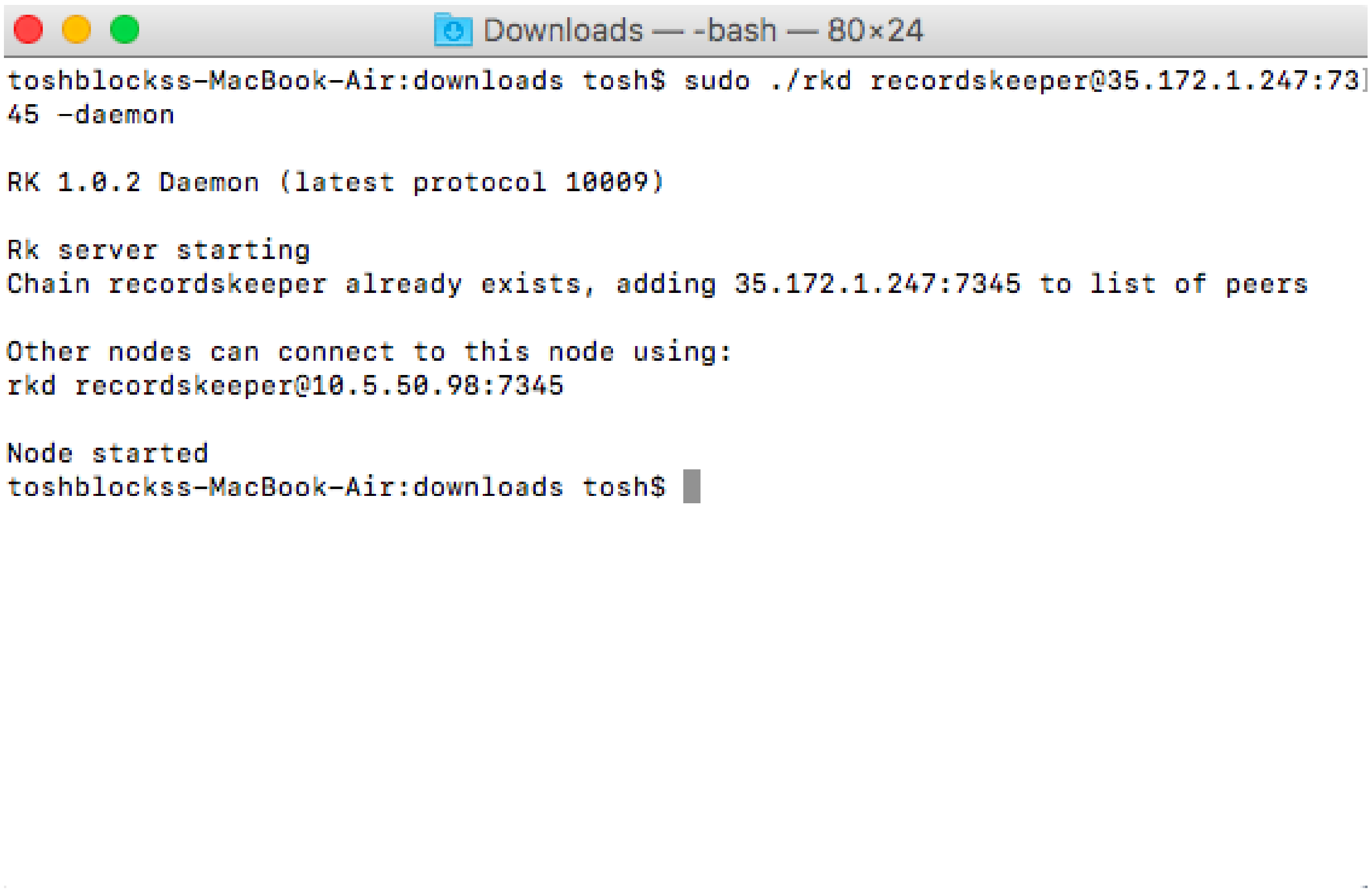
First, kill the daemon process, and then try connecting to the RecordsKeeper blockchain again. If the problem persists, restart your computer and then repeat the whole process of connecting to the RecordsKeeper blockchain again.

Note: *Mac users now go to the mining-permissions section

Mining Permissions

Connecting RecordsKeeper on Mac

You will see the following message on your Mac command line terminal after you execute the command to connect to the RecordsKeeper blockchain.



Downloads — -bash — 80×24

```
toshblockss-MacBook-Air:downloads tosh$ sudo ./rkd recordskeeper@35.172.1.247:73
45 -daemon

RK 1.0.2 Daemon (latest protocol 10009)

Rk server starting
Chain recordskeeper already exists, adding 35.172.1.247:7345 to list of peers

Other nodes can connect to this node using:
rkd recordskeeper@10.5.50.98:7345

Node started
toshblockss-MacBook-Air:downloads tosh$ 
```

RecordsKeeper Permissions

RecordsKeeper Testnet

The mining for RecordsKeeper Testnet is open to everyone, so when you connect to the RecordsKeeper Testnet, you will receive all the permissions for your default address

RecordsKeeper Mainnet

For Mainnet, when your node gets connected, you will receive the permissions to connect, send, and receive. Now, look for your default XRK address from the command given below, which will display your node's wallet address. This address is your "default XRK address" or "public address" of the RecordsKeeper blockchain in which you will receive XRK tokens. To check the address, run the following command:

```
./rk-cli recordskeeper getaddresses
```

Submit the following to receive Mining Permissions for RecordsKeeper Mainnet.

Note: Copy the above generated address and send it to us here .

Only after the RecordsKeeper team grant mining permissions to your node address will you be able to mine XRK tokens into your default address.

To retrieve the private key for your node address, run this command:

```
./rk-cli recordskeeper dumpprivkey {default_XRK_address}
```

Note: Please store this private key safely. Losing it will result in the loss of XRK tokens.

After completing the above process, you can check for your node's information (best block and synced block) by running the following commands:

```
./rk-cli recordskeeper getinfo
./rk-cli recordskeeper getblockchaininfo
```

Your node will sync with the best block, and then your node can start mining and your balance will get updated with the mined XRK tokens.

In case you have entered the wrong IP address, then it will report this error:

Warning: Error: Couldn't initialize permission database for blockchain recordskeeper. Probably rkd for this blockchain is already running. Exiting…

Please check the IP address and port properly to connect to the RecordsKeeper blockchain.

Note: If you have already created a wallet address and you want to add it as your miner address, then run this command from the command line terminal:

```
./rk-cli recordskeeper importprivkey {private_key}
```

Connecting to RecordsKeeper Blockchain after Permissions

Once the permissions for RecordsKeeper Mainnet have been granted, you can directly connect to the RecordsKeeper chain and see your mining progress. You can run the following commands to connect to the RecordsKeeper blockchain and view the mining address.

As the IP configuration was already stored with you when you initiated the connection, you can directly run the following command:

```
rkd recordskeeper -daemon
```

_static/MacRKAfterMining.jpg

You can run the getinfo command or getaddressbalances command to see the balance in your node or the node address.

```
rk-cli recordskeeper getinfo
```

_static/MacGetInfoMining.jpg

Note: You can view your balances in the balance output of the getinfo command.

OR

```
rk-cli recordskeeper getaddressbalances <Your Node Address Given for Mining>
```

`_static/MacAddressBalancesAfterMining.jpg`

Note: Please do not use the address specified above. This address is only available for the demo purpose.

Stopping Blockchain

RecordsKeeper Mainnet

If you want to stop your RecordsKeeper node, you can use the following command from your command line terminal:

```
./rk-cli recordskeeper stop
```

RecordsKeeper Testnet

If you want to stop your RecordsKeeper-test blockchain node, you can use the following command from your command line terminal:

```
./rk-cli recordskeeper-test stop
```

3.5 Getting Started with Postman

This tutorial requires you to setup a RecordsKeeper blockchain and run the APIs through the Postman client. If you have not done so already, please download and install RecordsKeeper on a server. If you are using the RecordsKeeper blockchain on Windows, please read the Installation of Windows notes to adapt the instructions below.

Download the latest RecordsKeeper build and put the downloaded files under the root folder. You can also run these commands from the folder where the files are present.

Download the Postman client from here.

3.5.1 Postman client Settings

Set the following parameters to request JSON RPC commands for RecordsKeeper.

Request-type

Select POST method in Postman client for using RecordsKeeper JSON RPC APIs.

Endpoint

Enter the following endpoint in the url:

```
35.170.155.89:8378/
```

Authorization

To setup authorization parameters, first select the type of Authentication to: **Basic Auth**. You have to enter the username and password for your node for authorization, which you can access through rk.conf built in your recordskeeper directory.

Linux (Ubuntu):

From your terminal run following commands:

```
cd ~/.rk/recordskeeper/
cat rk.conf
```

It will open the configuration file from where you can copy rpcuser and rpcpassword for your node. Enter these in username and password textbox of the postman client.

Windows:

Go to the directory:

```
AppData > Roaming > Rk > recordskeeper
```

Then look for the rk.conf file and open it in any text editor. It will display rpcuser and rpcpassword for your node. Enter these in username and password textbox of the postman client.

Headers

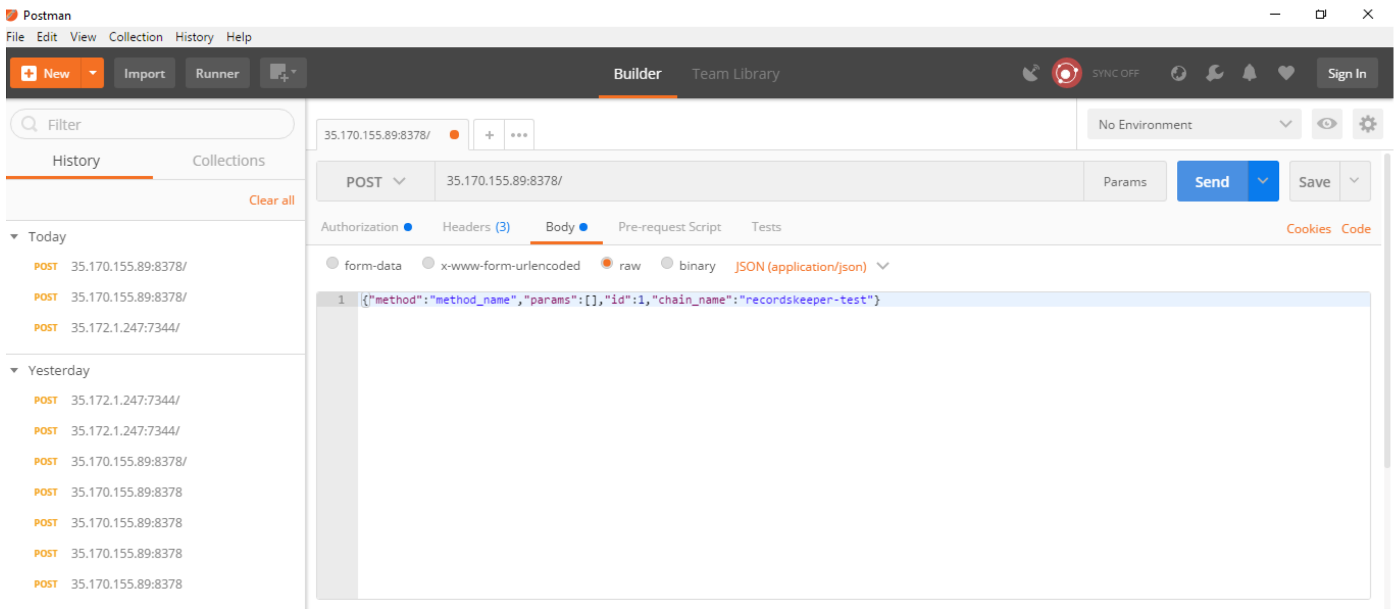
Headers are used to specify the metadata for the request type. It is a key-value based entry. Enter the following key-value entries in your header:

```
Content-type : application/json
Cache-control : no-cache
```

Body

Select the **Raw** type in your request. The Post API request will be in the following format, which is to be added inside the body of the request:

```
{"method":"method_name","params":[],"id":1,"chain_name":"recordskeeper-test"}
```



Today
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.172.1.247:7344/

Yesterday
POST 35.172.1.247:7344/
POST 35.172.1.247:7344/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378
POST 35.170.155.89:8378
POST 35.170.155.89:8378
POST 35.170.155.89:8378

POST 35.170.155.89:8378/

raw JSON (application/json)

```
{"method":"method_name","params":[],"id":1,"chain_name":"recordskeeper-test"}
```

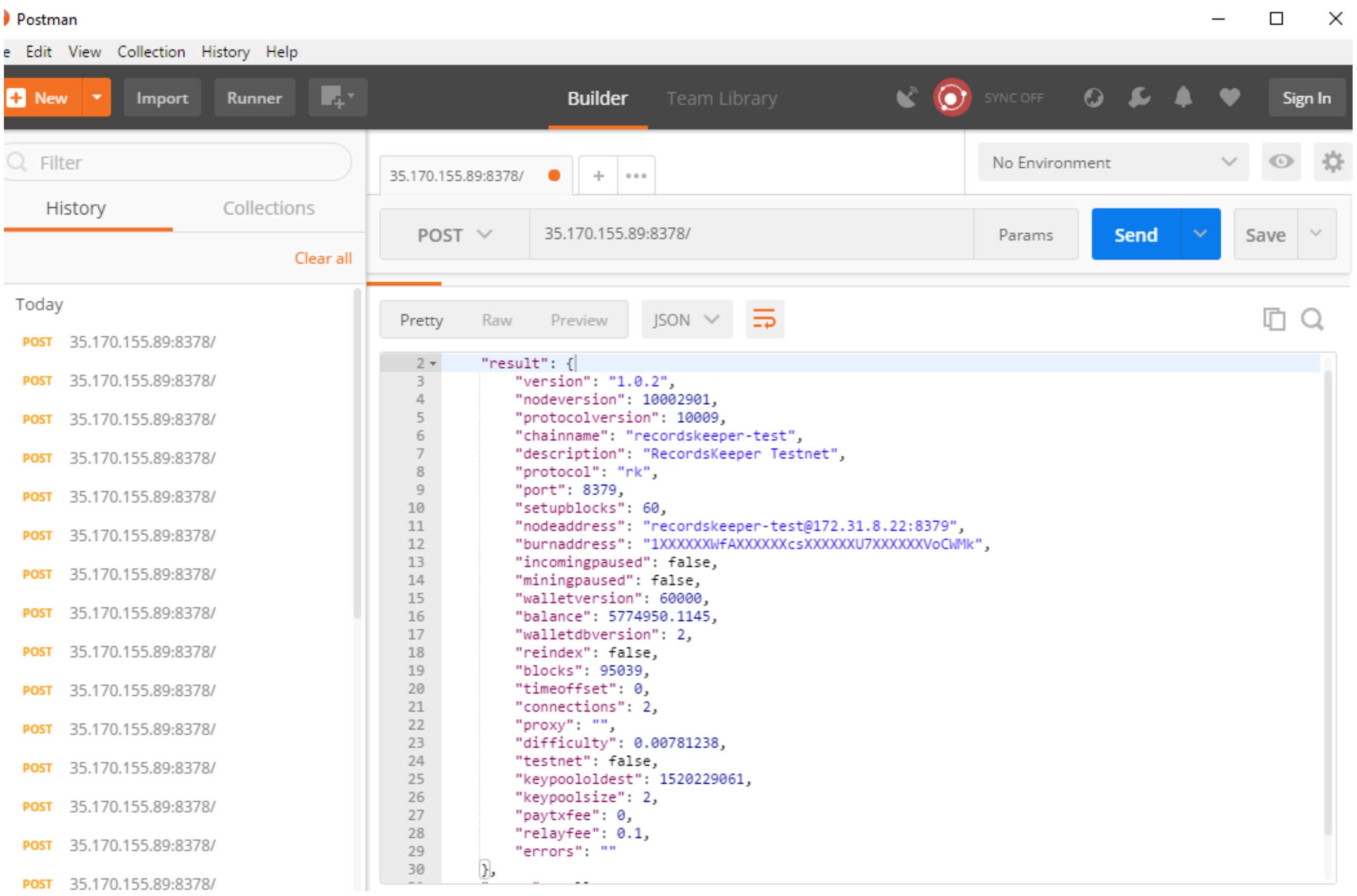
3.5.2 Running API Commands through Postman

Now your Postman client is set up and running, so you can use JSON RPC API commands to extract information, send transactions, and publish data over the RecordsKeeper node. Some of the important requests are as follows:

- Get general information about the RecordsKeeper node:

```
{"method":"getinfo","params":[],"id":1,"chain_name":"recordskeeper-test"}
```

The following result will be displayed:



POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/
POST 35.170.155.89:8378/

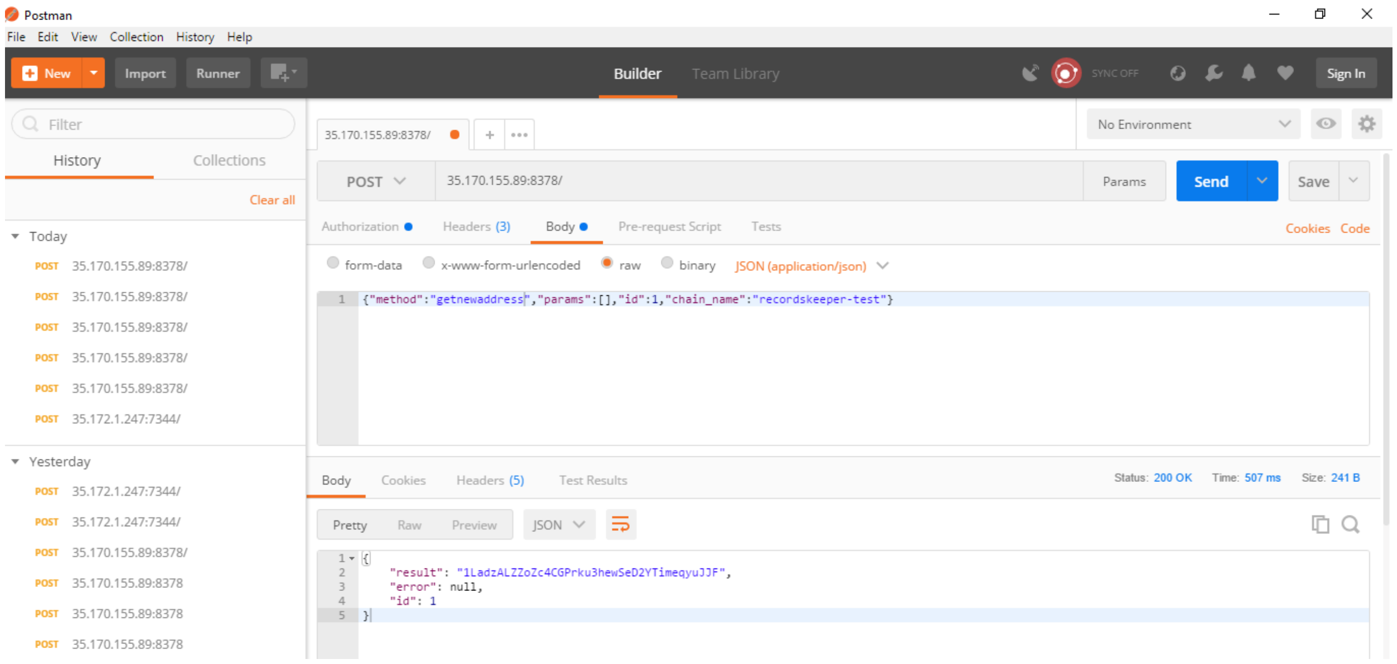
POST 35.170.155.89:8378/

```
    "result": {
        "version": "1.0.2",
        "nodeversion": 10002901,
        "protocolversion": 10009,
        "chainname": "recordskeeper-test",
        "description": "RecordsKeeper Testnet",
        "protocol": "rk",
        "port": 8379,
        "setupblocks": 60,
        "nodeaddress": "recordskeeper-test@172.31.8.22:8379",
        "burnaddress": "1XXXXXXWfAXXXXXXcsXXXXXXU7XXXXXXVoCWMk",
        "incomingpaused": false,
        "miningpaused": false,
        "walletversion": 60000,
        "balance": 5774950.1145,
        "walletdbversion": 2,
        "reindex": false,
        "blocks": 95039,
        "timeoffset": 0,
        "connections": 2,
        "proxy": "",
        "difficulty": 0.00781238,
        "testnet": false,
        "keypoololdest": 1520229061,
        "keypoolsize": 2,
        "paytxfee": 0,
        "relayfee": 0.1,
        "errors": ""
    },
```

- Create a new address in the RecordsKeeper Node wallet:

```
{"method":"getnewaddress","params":[],"id":1,"chain_name":"recordskeeper-test"}
```

The following result will be displayed:



POST 35.170.155.89:8378/

```
{"method":"getnewaddress","params":[],"id":1,"chain_name":"recordskeeper-test"}
```

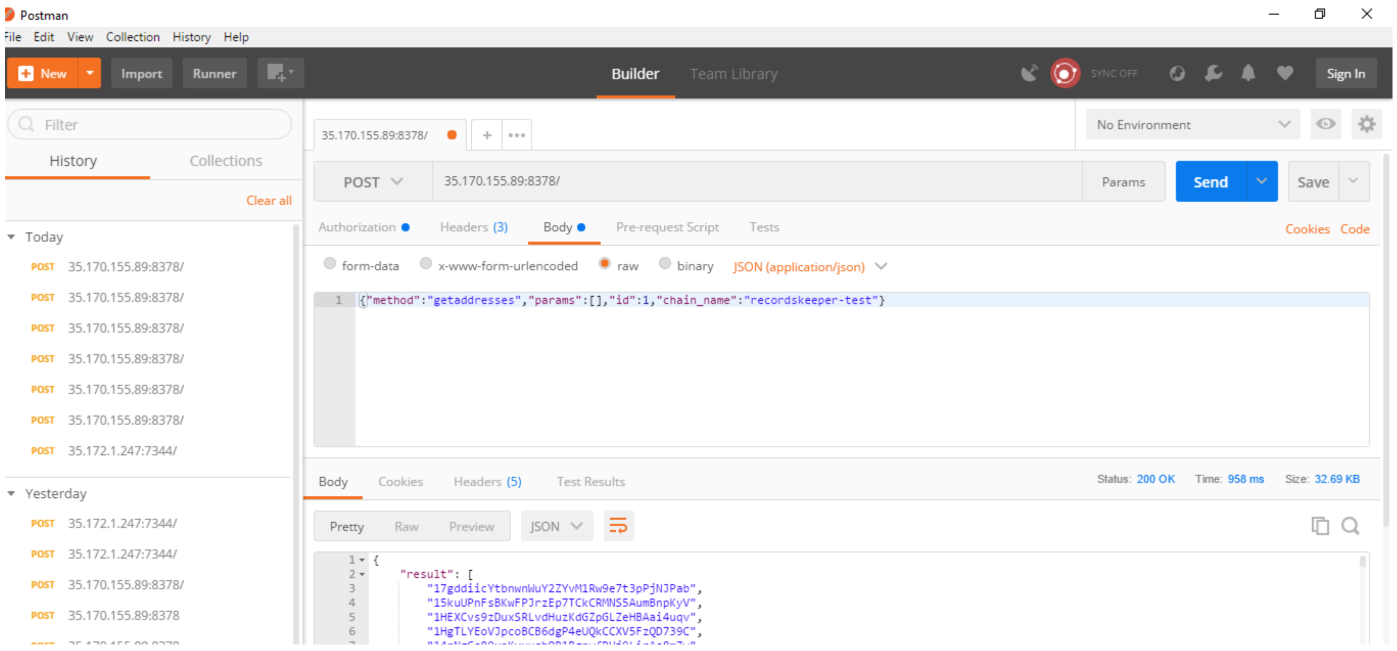
Status: 200 OK Time: 507 ms Size: 241 B

```
{
    "result": "1LadzALZZoZc4CGPrku3hewSeD2YTimeqyuJJF",
    "error": null,
    "id": 1
}
```

- List all addresses in the RecordsKeeper node wallet:

```
{"method":"getaddresses","params":[],"id":1,"chain_name":"recordskeeper-test"}
```

The following result will be displayed:



POST 35.170.155.89:8378/

```
{"method":"getaddresses","params":[],"id":1,"chain_name":"recordskeeper-test"}
```

Status: 200 OK Time: 958 ms Size: 32.69 KB

```
{
    "result": [
        "17gddiicYtbnwnWuY2ZYvM1Rw9e7t3pPjNJPab",
        "15kuUPnFsBKwFPJrzEp7TCkCRMNS5AumBnpKyV",
        "1HEXCvs9zDuxSRLvdHuzKdGZpGLZeHBAai4uqv",
        "1HgTLYEoVJpcoBCB6dgP4eUQkCCXV5FzQD739C",
```

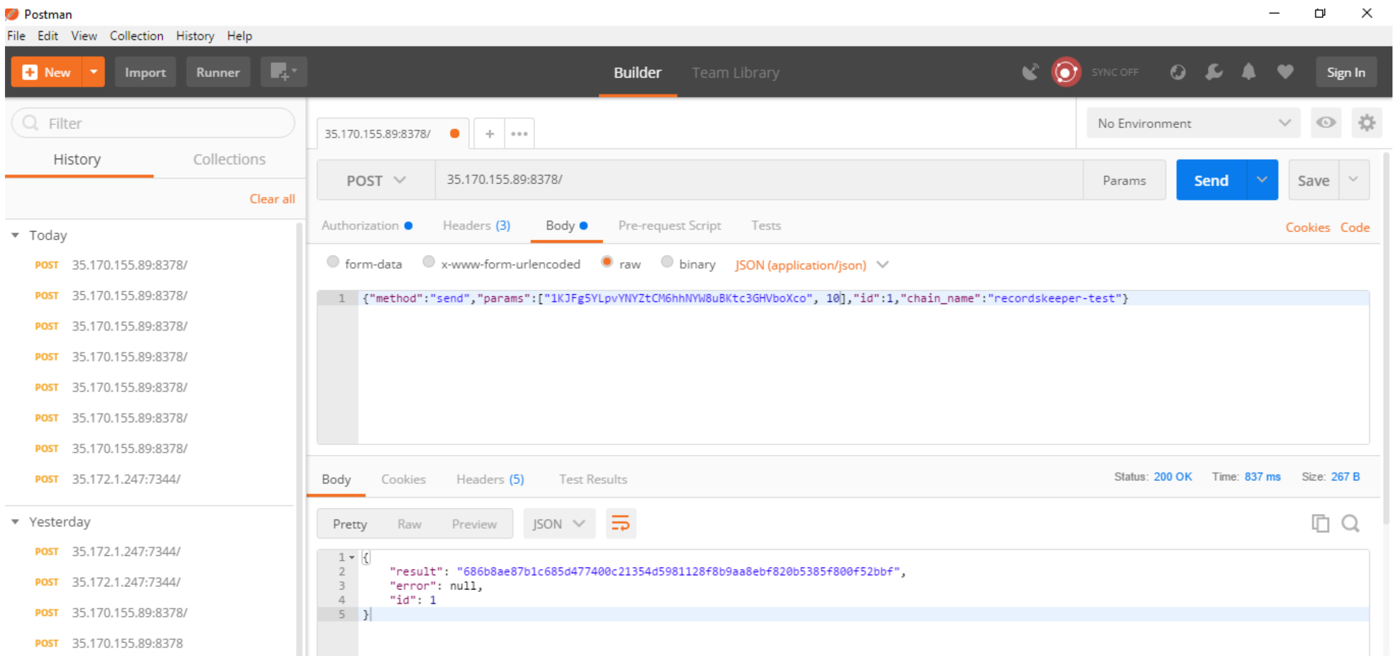
3.5.3 Sending a Transaction in RecordsKeeper

The RecordsKeeper blockchain works on the same backend as Bitcoin algorithms. Both the RecordsKeeper Testnet and Mainnet can be used to send and receive XRK tokens. Use the following API commands to send transactions on the RecordsKeeper blockchain.

Send

```
{"method":"send","params":["1KJFg5YLpvYNYZtCM6hhNYW8uBKtc3GHVboXco", 10],"id":1,
↪"chain_name":"recordskeeper-test"}
```

The following result will be displayed:



POST 35.170.155.89:8378/

Body: raw, JSON (application/json)

```
{"method":"send","params":["1KJFg5YLpvYNYZtCM6hhNYW8uBKtc3GHVboXco", 10],"id":1,"chain_name":"recordskeeper-test"}
```

Status: 200 OK Time: 837 ms Size: 267 B

JSON

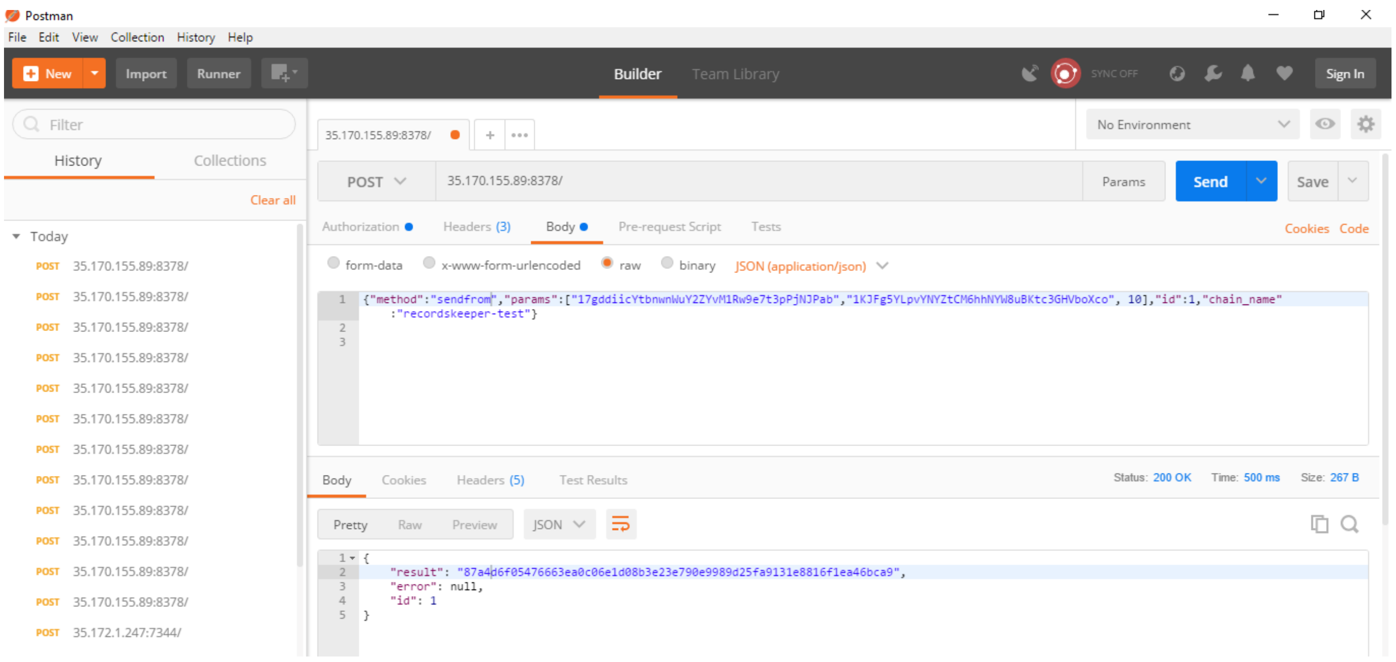
```
{
    "result": "686b8ae87b1c685d477400c21354d5981128f8b9aa8ebf820b5385f800f52bbf",
    "error": null,
    "id": 1
}
```

This command is used to send one or more XRK tokens to an address, returning the txid. The amount field is the quantity of XRK tokens and the address field is the address where you want to send the XRK tokens. This command will use the node's root address to send the transaction. Please make sure you have sufficient balance in the node's root address for transactions to propagate over the RecordsKeeper blockchain. You can also provide specific comments for the transaction, which are optional. The fees will be applied as per the transaction size.

Send from a Different Address

```
{"method":"sendfrom","params":["17gddiicYtbnwnWuY2ZYvM1Rw9e7t3pPjNJPab",
↪"1KJFg5YLpvYNYZtCM6hhNYW8uBKtc3GHVboXco", 10],"id":1,"chain_name":"recordskeeper-
↪test"}
```

The following result will be displayed:



```
{"method":"sendfrom","params":["17gddiicYtbnwnWuY2ZYvM1Rw9e7t3pPjNJPab","1KJFg5YLpvYNYZtCM6hhNYW8uBKtc3GHVboXco", 10],"id":1,"chain_name"
    :"recordskeeper-test"}
```

```
{
    "result": "87a4d6f05476663ea0c06e1d08b3e23e790e9989d25fa9131e8816f1ea46bca9",
    "error": null,
    "id": 1
}
```

This command is also used to send one or more XRK tokens to an address, returning the txid. Using this command you can specify the from-address which you want to use to send the transaction. The amount field is the quantity of XRK tokens, and the to-address field is the address where you want to send the XRK tokens. Please make sure you have sufficient balance in the from-address for transactions to propagate over the RecordsKeeper blockchain. The from-address used here is also one of the addresses generated for the node. You can also provide specific comments for the transaction which are optional. The fees will be applied as per the transaction size.

3.5.4 Publishing and Retrieving Data in RecordsKeeper

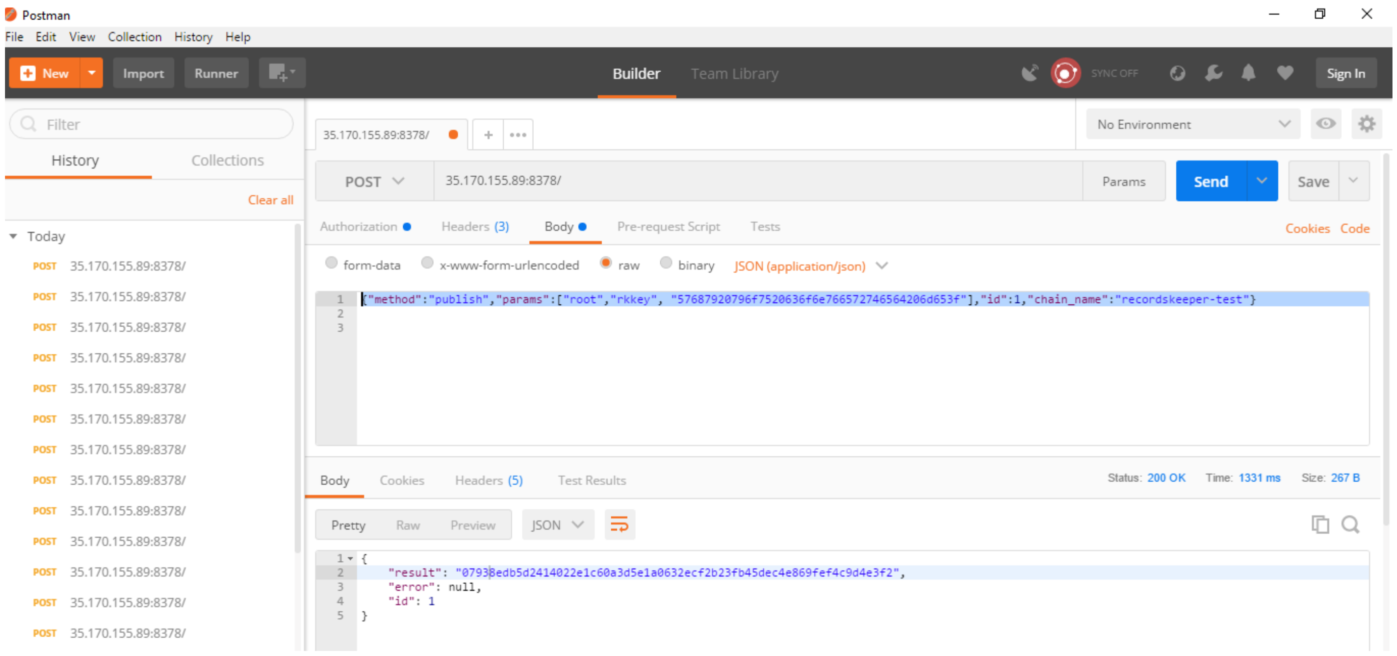
The RecordsKeeper blockchain is a public key-value based database on the blockchain. You can use the interactive command line to publish and retrieve stored information. As the blockahin is a shared concept, you can view all the published data and retrieve it using only a key or address. RecordsKeeper uses the streams to store the data. RecordsKeeper streams provide a natural abstraction for RecordsKeeper blockchain, which focus on general data retrieval, timestamping, and archiving, rather than the transfer of tokens between participants.

The "root" stream is open to all and anyone can publish data into the root stream. The following commands will give you a brief introduction about how to work with the data over RecordsKeeper blockchain.

Publish

```
{"method":"publish","params":["root","rkkey",
↪"57687920796f7520636f6e766572746564206d653f"],"id":1,"chain_name":"recordskeeper-
↪test"}
```

The following result will be displayed:



POST 35.170.155.89:8378/

```
{"method":"publish","params":["root","rkkey", "57687920796f7520636f6e766572746564206d653f"],"id":1,"chain_name":"recordskeeper-test"}
```

Status: 200 OK Time: 1331 ms Size: 267 B

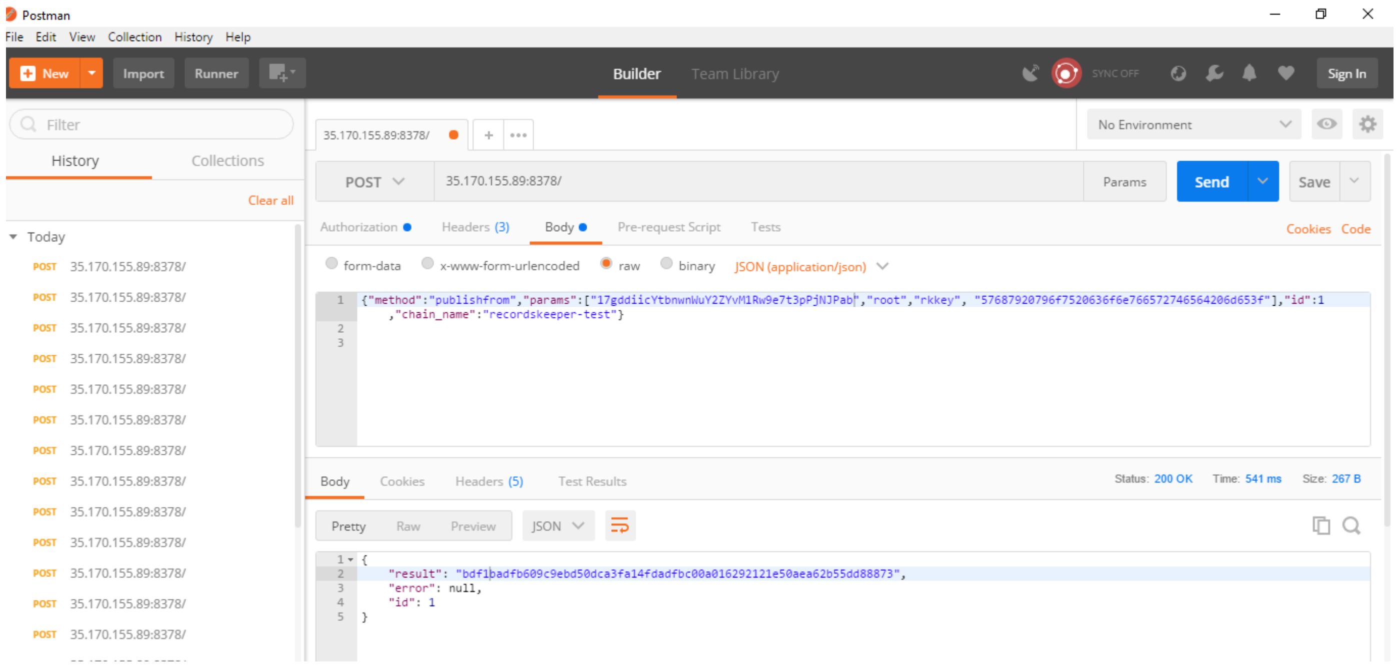
```
{
    "result": "07938edb5d2414022e1c60a3d5e1a0632ecf2b23fb45dec4e869fef4c9d4e3f2",
    "error": null,
    "id": 1
}
```

This command publishes an item in a stream. The stream name is passed, with the key provided in text form and a data-hex in hexadecimal format. It returns ref or creation txid. The data is published using the node's address. Use the next command discussed below to publish data from different address. The mining fees are applied as per the transaction size.

Publish from a Different Address

```
{"method":"publishfrom","params":["17gddiicYtbnwnWuY2ZYvM1Rw9e7t3pPjNJPab","root",
↪"rkkey", "57687920796f7520636f6e766572746564206d653f"],"id":1,"chain_name":
↪"recordskeeper-test"}
```

The following result will be displayed:



POST 35.170.155.89:8378/

```
{"method":"publishfrom","params":["17gddiicYtbnwnWuY2ZYvM1Rw9e7t3pPjNJPab","root","rkkey", "57687920796f7520636f6e766572746564206d653f"],"id":1
    ,"chain_name":"recordskeeper-test"}
```

Status: 200 OK Time: 541 ms Size: 267 B

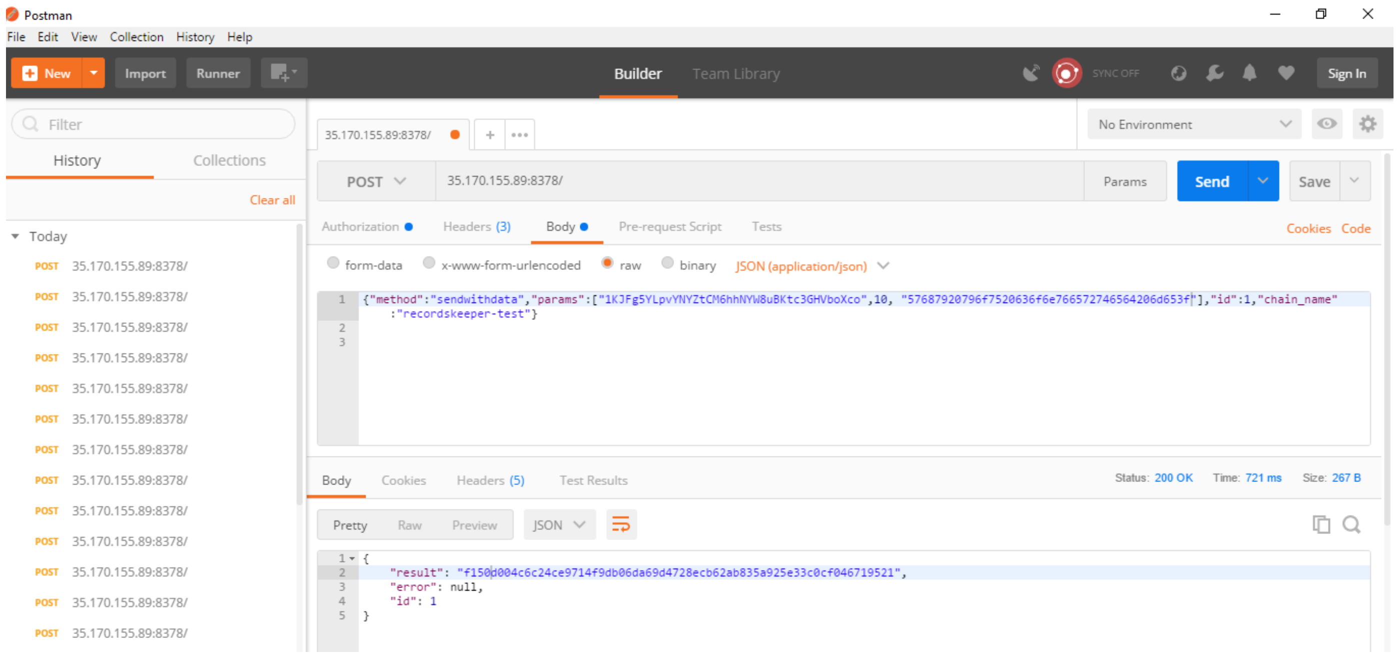
```
{
    "result": "bdf1badfb609c9ebd50dca3fa14fdadfbc00a016292121e50aea62b55dd88873",
    "error": null,
    "id": 1
}
```

This command works like publish, but publishes the item from the from-address. It is useful if the node has multiple addresses with different amounts. The mining fees are applied as per the transaction size.

Send as a Transaction

```
{"method":"sendwithdata","params":["1KJFg5YLpvYNYZtCM6hhNYW8uBKtc3GHVboXco",10,
↪"57687920796f7520636f6e766572746564206d653f"],"id":1,"chain_name":"recordskeeper-
↪test"}
```

The following result will be displayed:



POST 35.170.155.89:8378/

```
{"method":"sendwithdata","params":["1KJFg5YLpvYNYZtCM6hhNYW8uBKtc3GHVboXco",10, "57687920796f7520636f6e766572746564206d653f"],"id":1,"chain_name"
    :"recordskeeper-test"}
```

Status: 200 OK Time: 721 ms Size: 267 B

```
{
    "result": "f150d004c6c24ce9714f9db06da69d4728ecb62ab835a925e33c0cf046719521",
    "error": null,
    "id": 1
}
```

This works similarly to send, but with an additional data-only transaction output. You can pass raw data as data-hex hexadecimal string. It is also used to publish the data to a stream, pass an object like this {"for":StreamName,"key":"KeyName","data":"DataHex"} where stream is a stream name, ref, or creation txid, the key is in text form, and the data is hexadecimal. You can pass the amount as 0, if you are only using this to publish the data over the RecordsKeeper stream. You can also send some XRK tokens while publishing the data over the stream. The fees will be applied as per the transaction size.

Note: The address displayed here is a demo address. Please don't use this address in your transactions. You can create a new wallet or address by using the RecordsKeeper Wallet

3.6 Use Cases

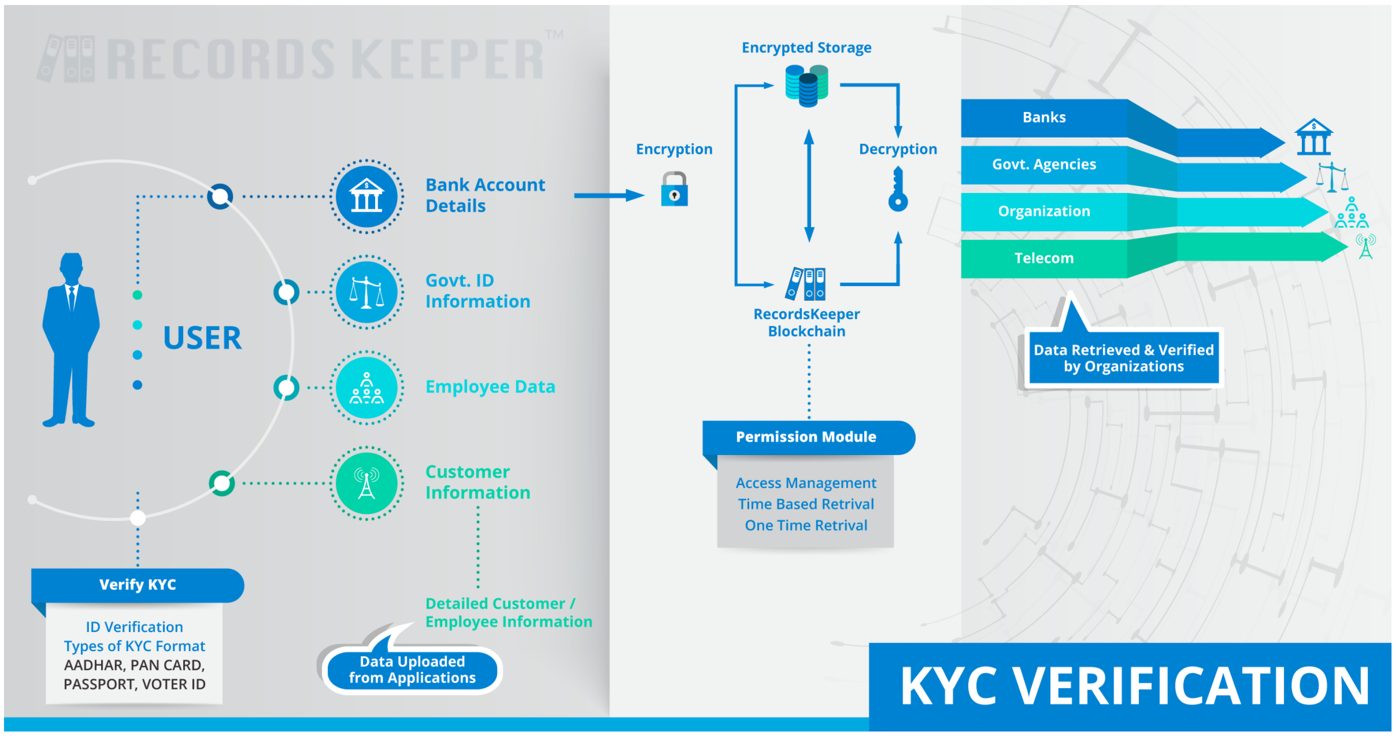
The following use-cases are provided for business and individual needs:

- *Enterprise KYC*
- *Insurance Records Keeping*
- *Corporate Compliances Records Keeping*
- *Verify Academic Certifications*
- *Supply Chain Management*
- *Trustless File Sharing*
- *Government Regulations*
- *Land Ownership Record Keeping*
- *Employee Verification*
- *Health Record Manifest In Judiciary*

3.6.1 Enterprise KYC

Storing and Verifying Enterprise KYC/Government IDs on the RecordsKeeper blockchain

KYC procedures and compliance have become increasingly important in the banking industry as regulators are keeping track of banks by noting down with whom they are doing business. This is done to prevent potential money laundering or terrorist financing. Currently, institutions offering financial or professional services are obliged to follow time-consuming and expensive practices for each new customer. These KYC processes can delay business as it can take between 30 to 50 days to complete satisfactorily. Some estimates show that global spending on AML compliance alone is close to $10 billion. Banks are also coming under pressure from various investors and analysts to reduce costs, but many expect the compliance budgets to increase in the coming years rather than decrease.

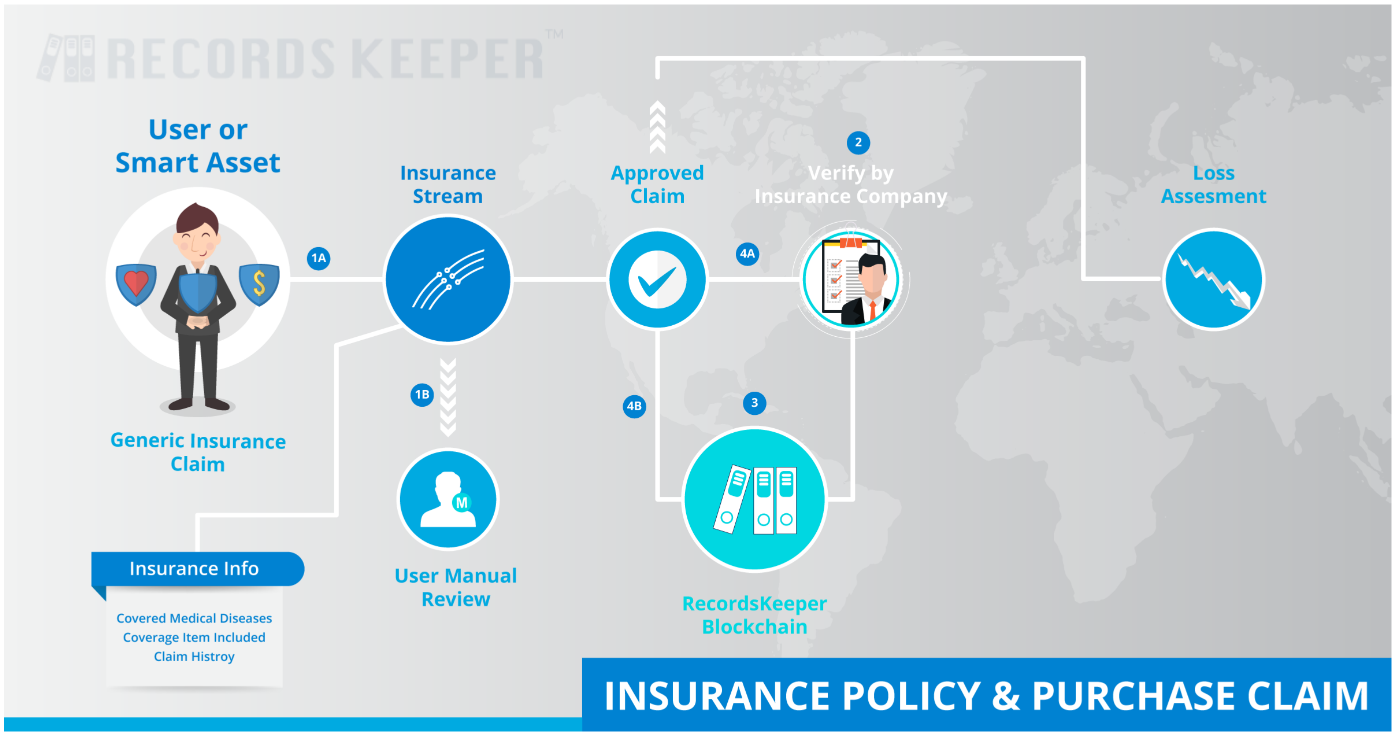


The adoption of blockchain technology could lead to the reduction of AML and KYC costs, thanks to its cross-institution client verification capability, as well as its effectiveness in monitoring and analyzing data required for AML and KYC checks. RecordsKeeper performs this use case flawlessly, without the need to maintain physical documents. The authenticity of these documents can be easily verified because of the nature of the blockchain. RecordsKeeper streams can maintain records of various identification cards, which can be shared and verified when required by any enterprise unit or government agency.

On the RecordsKeeper blockchain, the verification of a client or the legality of a transaction can be determined just once, with the final result cryptographically stored on the RecordsKeeper blockchain. Organizations such as banks or administration services providers would be able to get rid of the multistep AML and KYC processes. More specifically, all of the information related to a client would become available to organizations with the appropriate permissions via a distributed database that would be considered a single source of "truth." So, when a bank has verified a new client, they can then put the client's data on RecordsKeeper blockchain which can then also be accessed by other banks and accredited organizations, such as loan providers and insurance companies, without the need for the KYC process to be repeated by another party. These parties would know that the client's information has been independently audited and verified by various independent miners so that no more KYC checks would be necessary.

3.6.2 Insurance Records Keeping

Using RecordsKeeper Blockchain to store insurance records:



This use case addresses improvements in the operational functions of an insurance company. As per the traditional procedure is shown in the image, when a user makes a general insurance claim, all the insurance information gets verified from the insurance stream manually. If the information is correct, the insurance claim is verified by the insurance company. However, this procedure has the following problems:

1. Data Security
2. Fraudulent information
3. Trust issues

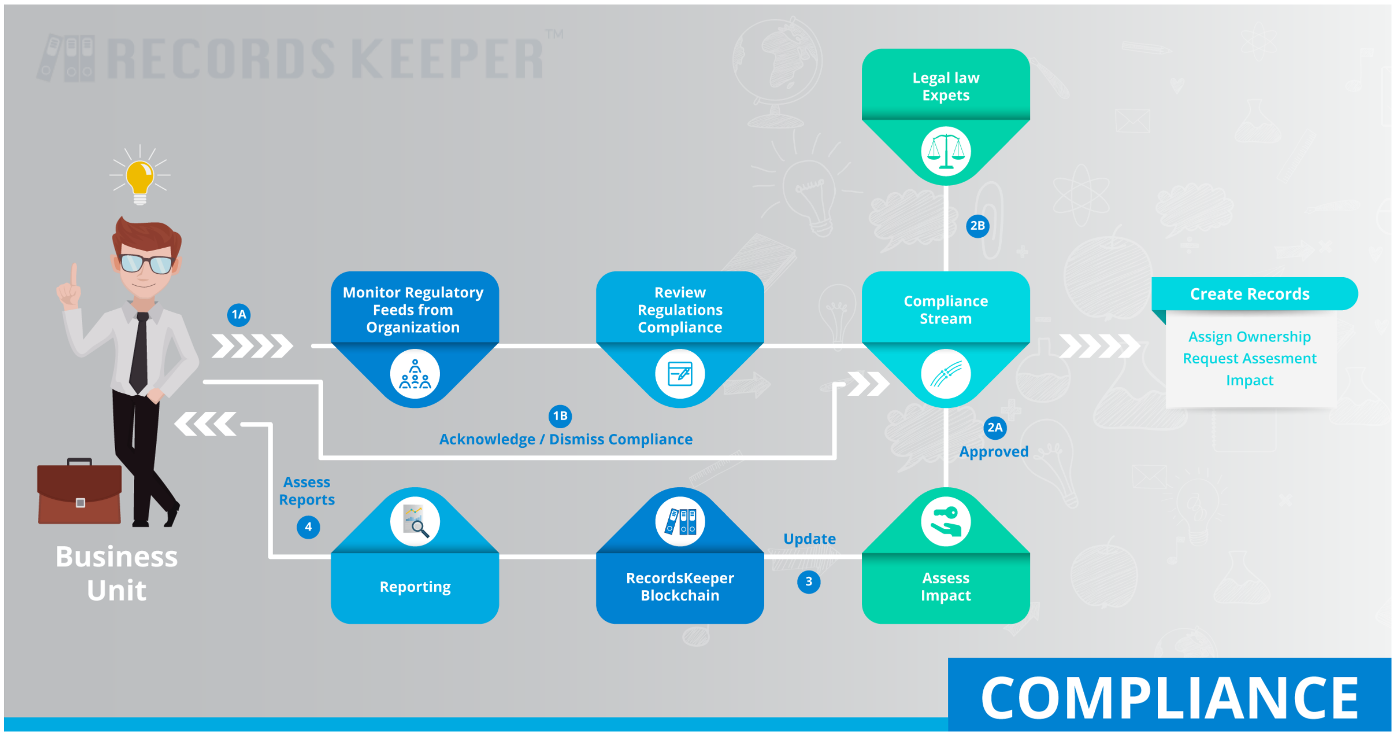
RecordsKeeper can remove all of these problems. RecordsKeeper is a public blockchain for records keeping and data security. Let's discuss each problem individually:

1. **Data Security**: Data stored in RecordsKeeper is decentralized so there is considerably less possibility of data loss as compared to the present centralized system because a copy of the data is present in every node which is connected to the network. This way, the data can be quickly recovered in the event of data loss.
2. **Fraudulent information**: Unlike with the traditional insurance record keeping system, data stored on the RecordsKeeper blockchain cannot be tampered with or modified. Moreover, insurance companies can easily identify false applications, so there is less chance of fraud.
3. **Trust issues**: For every insurance company, it is imperative to gain the client's trust. In the traditional process, there are many chances for fraud, so it is not easy for a client to trust the insurance company, but with the help of RecordsKeeper blockchain, trust issues can be resolved.

3.6.3 Corporate Compliances Records Keeping

Using RecordsKeeper Blockchain to store Corporate Compliances

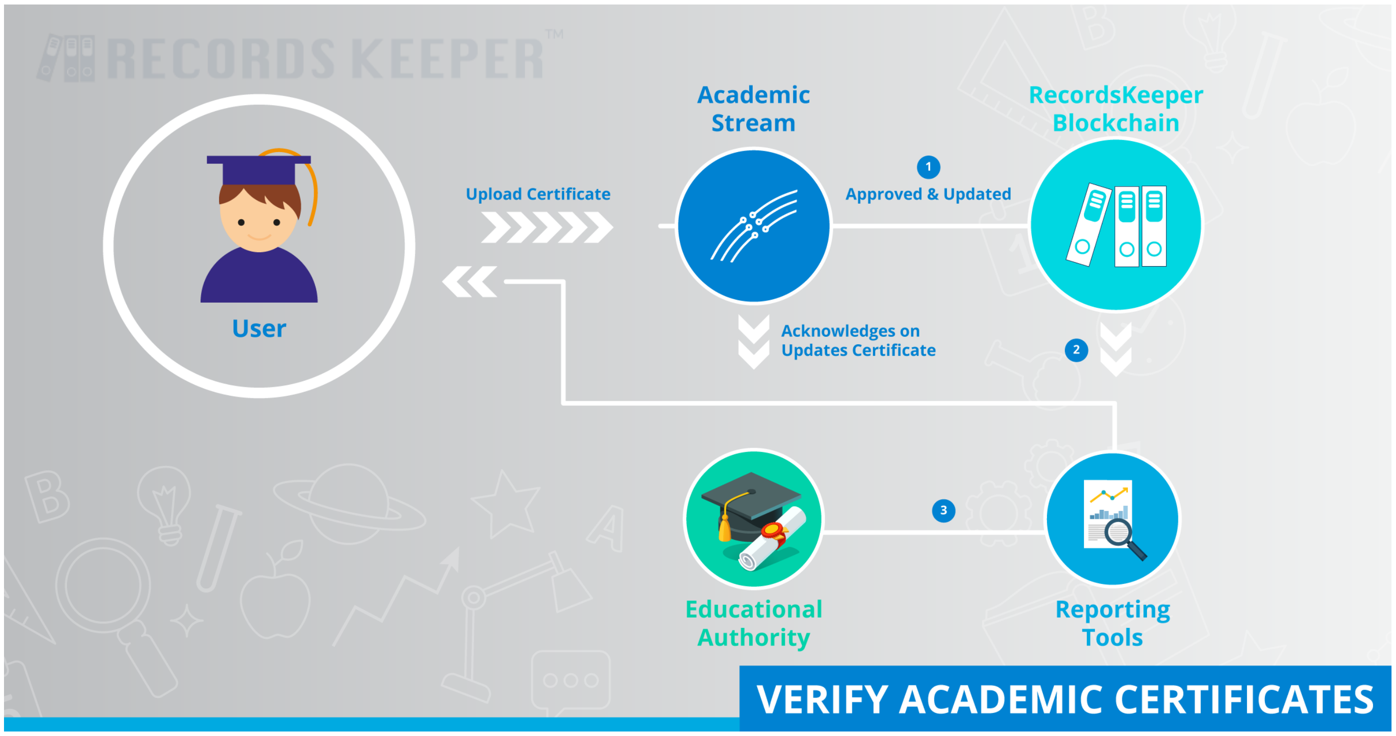
One of the most admired features of blockchain technology, from a compliance perspective, is its practical immutability. As soon as data is saved onto the chain, it cannot be modified or removed. This is one of the major features enabling blockchain to be used for the transfer of any digital asset.



RecordsKeeper avoids pain points for financial institutions and regulators. Saving one shared permanent record on the blockchain reduces the need for duplication, which can represent tremendous savings for the entire industry. It would also expedite the regulatory review process since there would no longer be a need for reconciliation. A business unit can monitor regulatory feeds and update their records based on regulations in the RecordsKeeper blockchain. Each compliance document is either acknowledged or dismissed based on the regulations. Every approved document is stored in the RecordsKeeper ledger and can be shared and verified as required.

3.6.4 Verify Academic Certifications

Upload and Verify Academic Certifications over RecordsKeeper



Academic certificates can be easily issued and verified using the RecordsKeeper ledger. Educational authorities can issue a certificate and give the user a receipt which they can share with any third party to prove the authenticity of the certificate. When the third party receives this receipt, they can easily check its authenticity on the RecordsKeeper ledger.

Additionally, RecordsKeeper provides the following features:

Transparency – Both the parties who are interested in viewing academic credentials can see them on the RecordsKeeper blockchain. This ensures that only people with ownership rights can make decisions about who has access to this information.

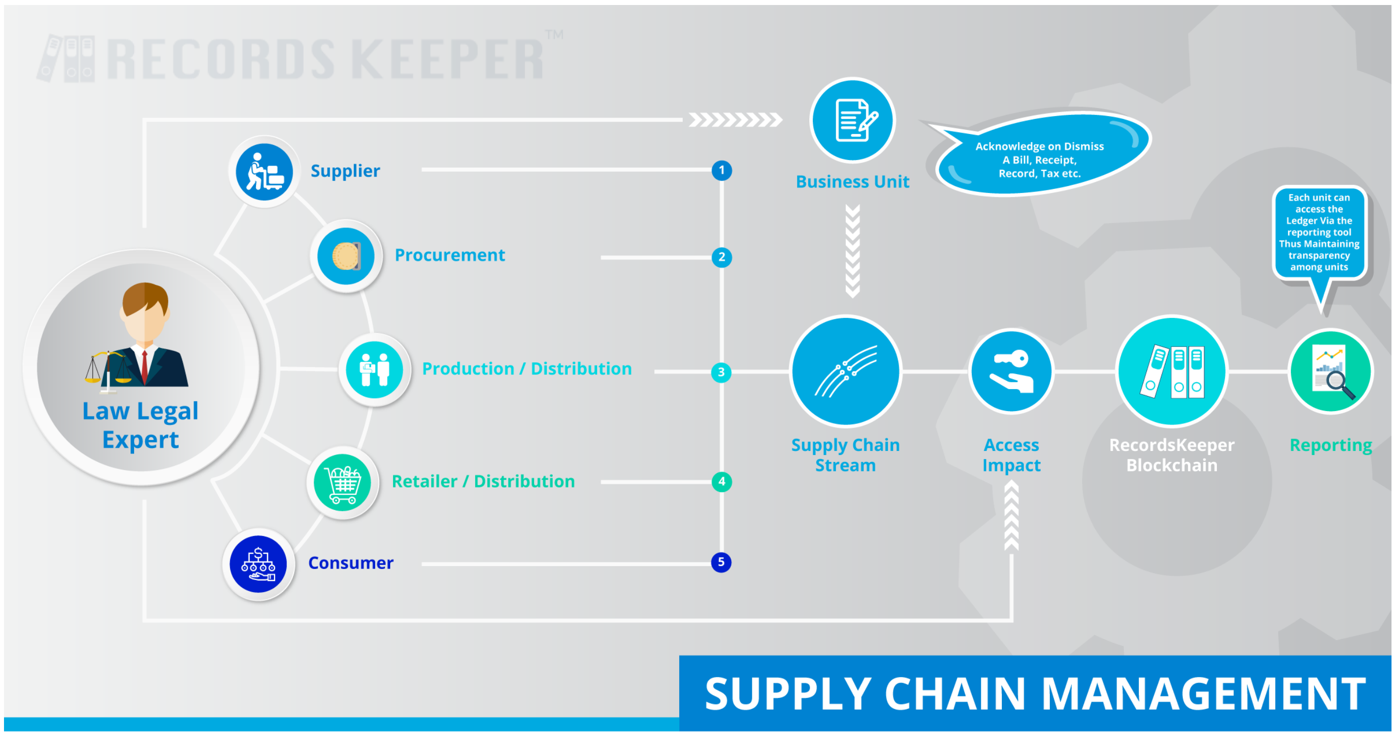
Immutability – Blockchains are the most secure source for storing the information right now. They rely on the integrity of the network to ensure the authenticity of the stored information. Thus, academic certificates stored on the RecordsKeeper blockchain are immutable.

Disintermediation – Using the RecordsKeeper blockchain to store and share academic credentials helps us bypass the need for a central controlling authority that manages and keeps records. This makes the overall process of storing credentials more trustworthy as there are no middlemen involved.

Collaboration – Once the information becomes available on the RecordsKeeper blockchain, it is much easier to ascribe ownership, and therefore safer to share the information without the fear of this information being compromised.

3.6.5 Supply Chain Management

Supply Chain Management over the RecordsKeeper Blockchain



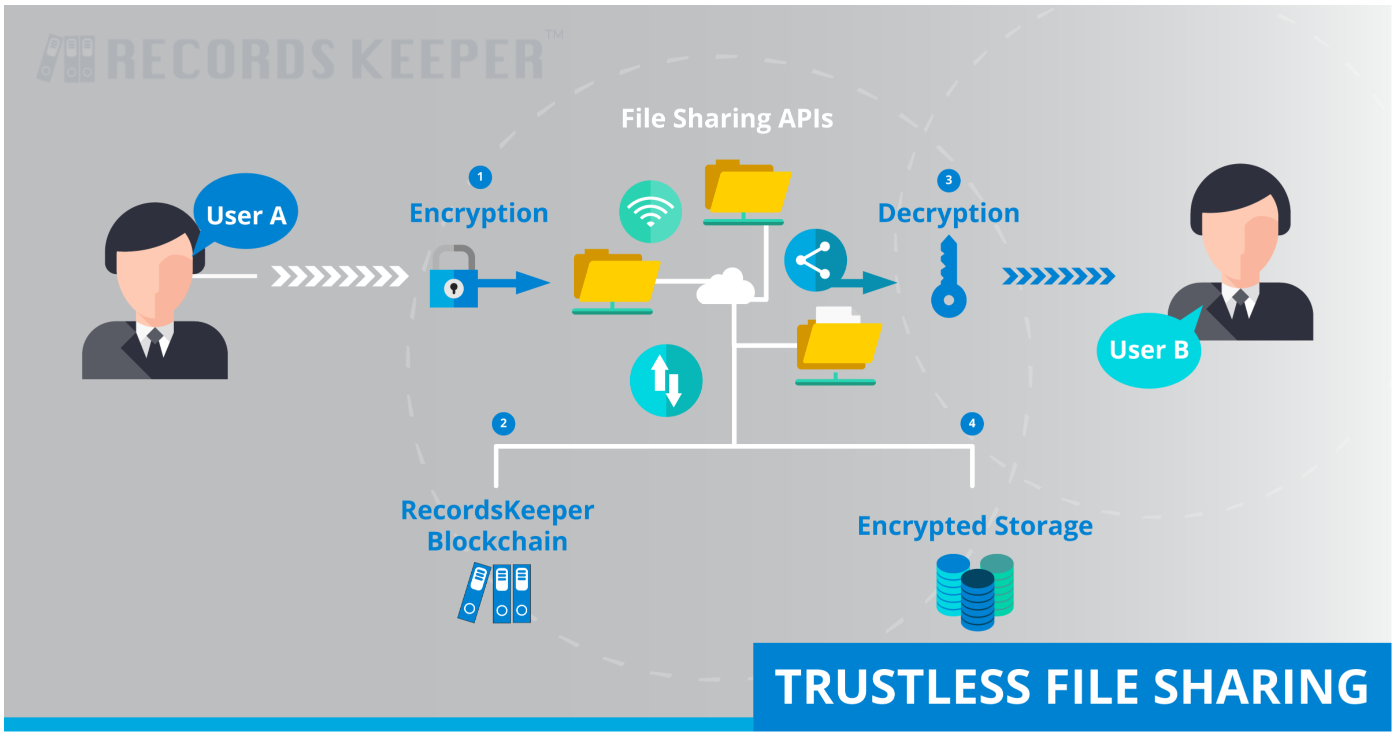
RecordsKeeper can help improve the supply chain management in the following ways:

- Recording the quantity and transfer of assets – such as pallets, trailers, or containers, etc. – as they move between supply chain nodes
- Tracking purchase orders, change orders, receipts, shipment notifications, or other trade-related documents
- Assigning or verifying certifications or certain properties of physical products; for example, determining if a food product is organic or fair trade
- Linking physical goods to serial numbers, barcodes, digital tags such as RFID, etc.
- Sharing information about the manufacturing process, assembly, delivery, and maintenance of products with suppliers and vendors.

Each unit has a set of compliance conditions which need to be followed. RecordsKeeper can make sure that these are being followed, throughout the process. For each unit, the current status of the supply and its related compliance is approved and recorded in the ledger by the unit itself, thus helping to maintain transparency among all units. RecordsKeeper will store copies of receipts, orders, or notifications, etc. and will make sure that their integrity is maintained. RecordsKeeper helps businesses set the rules for recording the data so that they can set the compliance conditions themselves. Only when the conditions are met does RecordsKeeper allow the user to share the data.

3.6.6 Trustless File Sharing

Trustless File Storing and Sharing over RecordsKeeper Blockchain

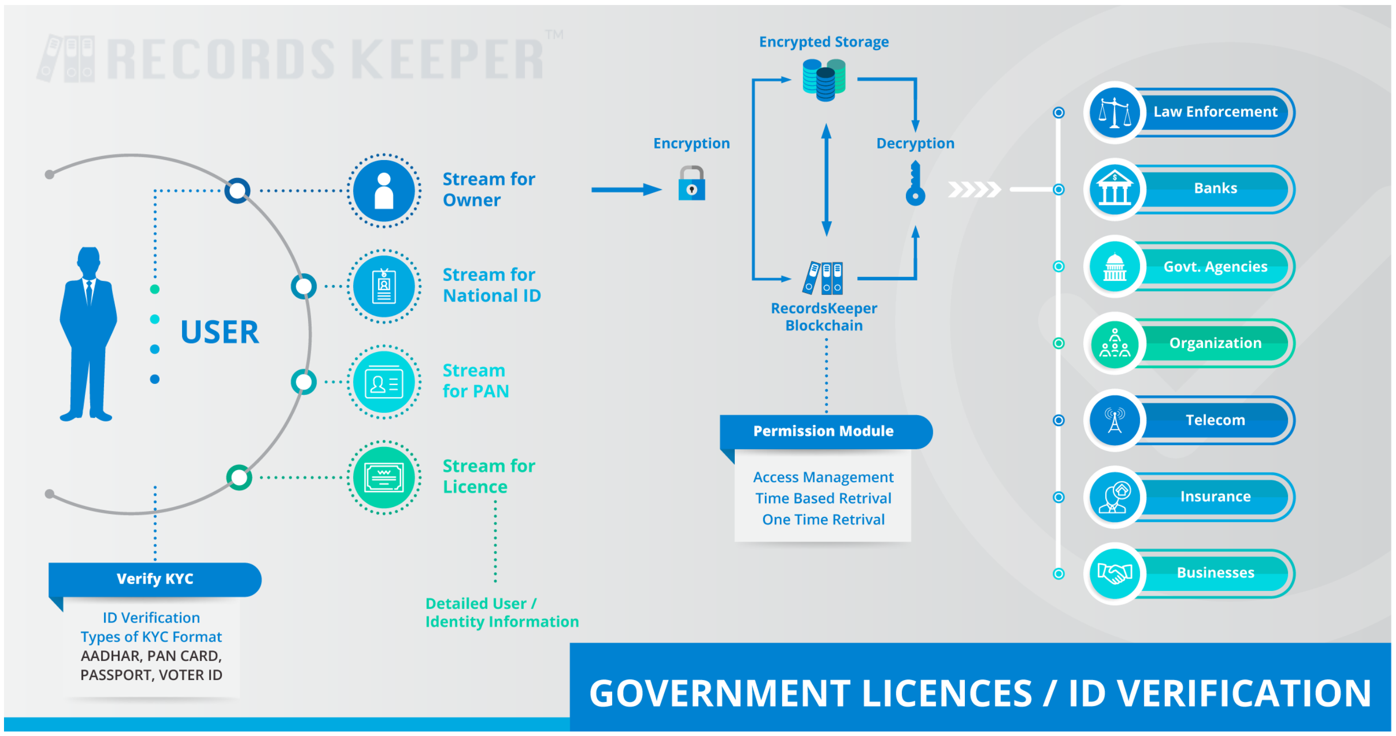


Data sharing has always been prone to attacks, hackers are always present in a search for ways to hack any data by means of channel diversion or accessing the encrypted transferred data. RecordsKeeper can be used for creating a system which will help in sharing data over the network securely. This system will encrypt the file with receiver's public key before sharing. Once the receiver receives the file, they can decrypt it using their private keys. At each stage, the Recordskeeper will maintain the metadata of the file and keep track of all the changes made in it. Thus, both the parties can verify if a document has been manipulated or not.

The primary benefit of using RecordsKeeper Blockchain for file sharing is the safety of the data. Information present on RecordsKeeper Blockchain is immutable and unalterable which makes it safe and secure. In case of the RecordsKeeper Blockchain, if a user uploads a file then the hash of the file gets stored in the Blockchain and then the user can verify the authenticity of the file by using the hash and claim the ownership.

3.6.7 Government Regulations

Government Regulation Policies on the RecordsKeeper Blockchain

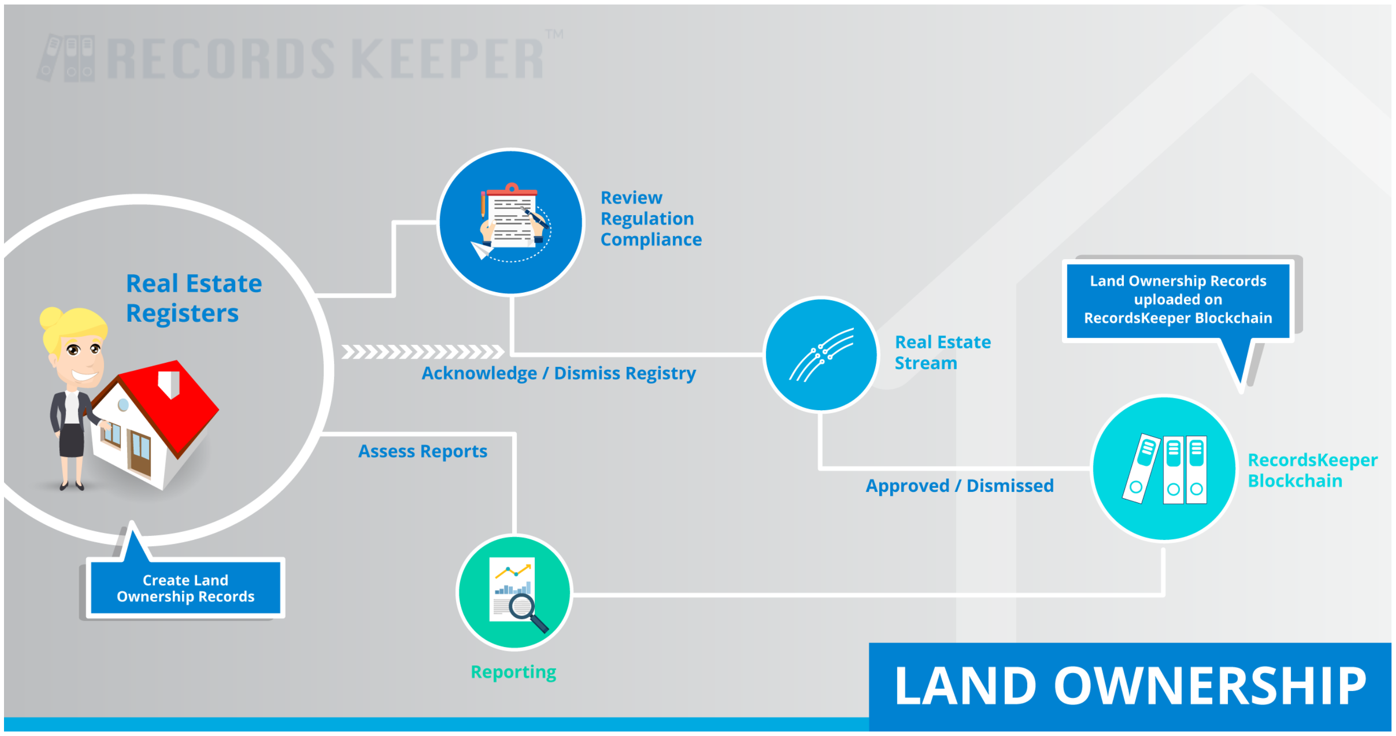


Identification documents are a tedious but necessary responsibility for all citizens. These documents can be easily faked, which has a direct impact on many government agendas; for example, governments are often not able to implement health or food policies efficiently due to partial or misleading information. RecordsKeeper can be effectively used as a solution to this problem; records with proper and unalterable information can be saved in the RecordsKeeper ledger, which can be easily accessed when required.

This information is safe with RecordsKeeper as the details of the document are saved in the ledger, making the verification process effortless. The RecordsKeeper platform aims to significantly reduce the amount of time and resources spent on cross verification of identification documents. Once the data is stored on the RecordsKeeper blockchain with the document hash, users can claim ownership of the files. By this mechanism, the risk of fake document creation is significantly reduced. Thus, with the help of RecordsKeeper, we can increase the authenticity of government identification documents.

3.6.8 Land Ownership Record Keeping

Land Ownership verification on the RecordsKeeper Blockchain


RECORDS KEEPER
Real Estate Registers
Create Land Ownership Records
Review Regulation Compliance
Acknowledge / Dismiss Registry
Assess Reports
Reporting
Real Estate Stream
Approved / Dismissed
Land Ownership Records uploaded on RecordsKeeper Blockchain
RecordsKeeper Blockchain
LAND OWNERSHIP

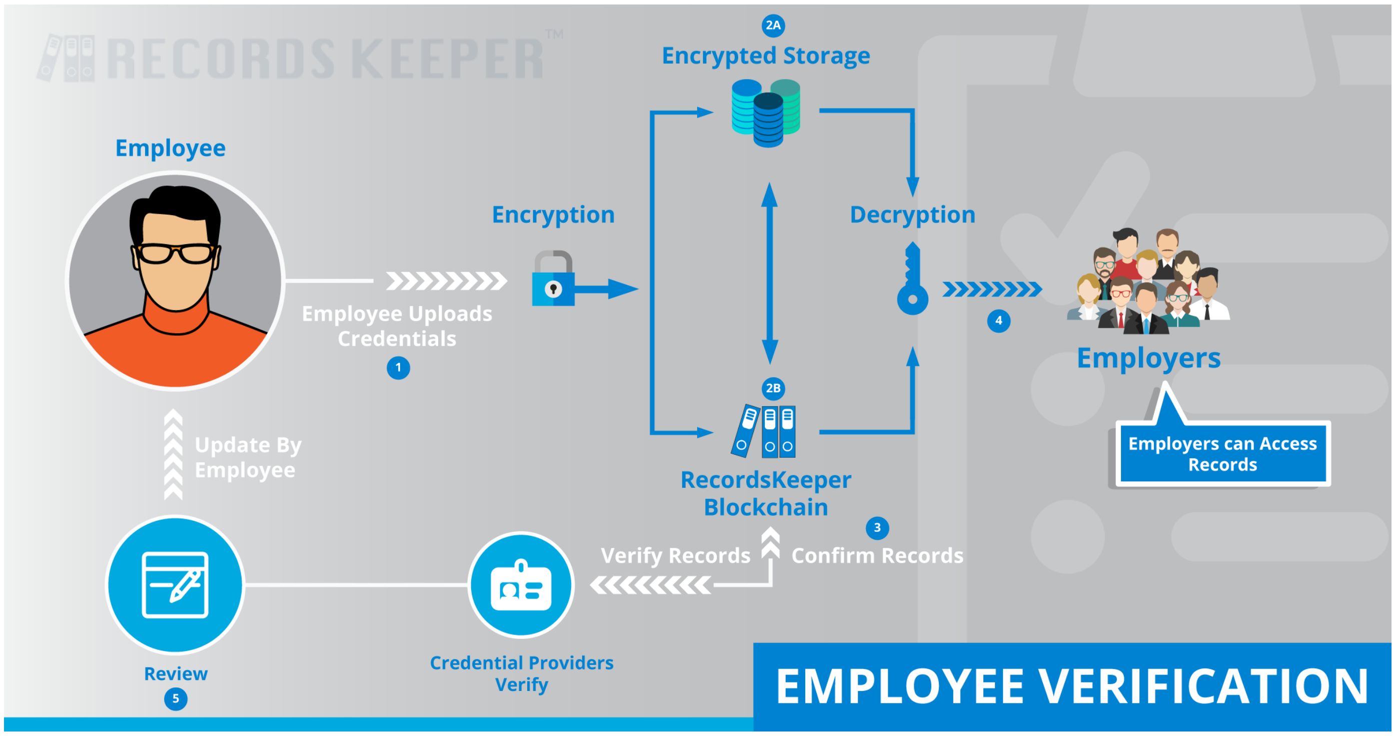
For any high-value property (such as real estate, cars, or artwork) it is essential to have accurate records which can identify the owner. These records are used to protect owners' rights in various situations: for example, in the case of theft, in resolving disputes, or in the transfer of ownership after a sale. Thus, it is necessary to maintain not only the accuracy but also the completeness of this information to prevent unauthorized and fraudulent actions.

Currently, people have to rely on a trusted third party. For example, a government agency might be responsible for keeping track of ownership information. Sometimes, these records are not preserved systematically. RecordsKeeper solves this problem entirely, as it provides a way of sanctioning, approving, and saving all sorts of asset information which can never be falsified or altered.

The current process for clearing a land deed is very complicated and requires house buyers to hire an intermediary to clear the land deed for them. This is a complete waste of resources, both human and monetary, to accomplish something that can easily be done with the help of a public blockchain such as RecordsKeeper's. In addition to making the process a whole lot more efficient, registering land deeds on a public blockchain also ensures that owner information cannot be manipulated in any way, adding another layer of security for homeowners.

3.6.9 Employee Verification

Employee Credential Verification on the RecordsKeeper Blockchain

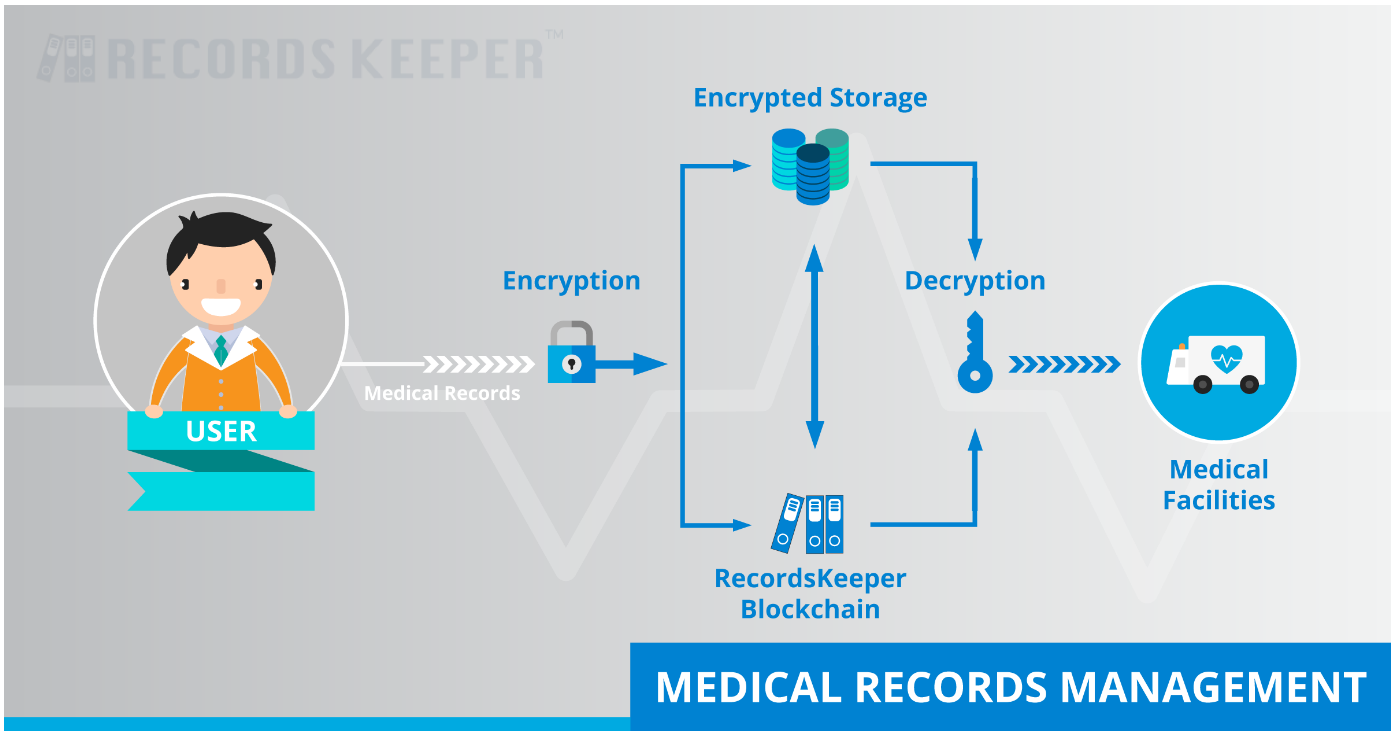


Employers face a major hurdle in hiring employees when they require verification of candidates' credentials. A survey conducted by one of the largest online job finder sites, CareerBuilder, found that a staggering 58% of employers have found fraudulent details on the resumes of individuals. A separate report on the 2015 hiring outlook done by HireRight, a company offering global background checks, employment verification services, and drug testing shows that screening of misrepresented resumes is required for 86% of employers. There is no doubt that resume accuracy is a common problem in the current environment and the effort required to verify credentials impacts the hiring process and costs employers time and money. This is where RecordsKeeper will increase transparency and address fraud in employee credentials.

RecordsKeeper helps employers in verifying employee details, by allowing for the inspection of the source of their funds, business interests, and employment history. They can also monitor the verification progress along the way. Every employer has to perform the KYC process individually and upload the validated information and documents to RecordsKeeper where it is stored as digitized data and tagged with a unique identification number for each customer. By using this reference number, the employer can access the stored data to perform due diligence whenever employees apply for a new job.

3.6.10 Health Record Manifest In Judiciary

Health Record Manifest requirements for Judiciary on the RecordsKeeper Blockchain



In today's digital society, everyone is concerned about the privacy and security of their data, especially in health care. So we need more transparency in medical information to protect patients' privacy and reduce possible security breaches. People nowadays are faking their health status and problems to claim health insurance while some also fake medical cases to claim fraudulent benefits. These cases, when taken to court, become complicated due to the lack of judicial system assertions. RecordsKeeper enables the sharing of information on a secure, tamper-proof, and immutable platform. Nothing can be tampered with or fraudulently conveyed in the RecordsKeeper ledger. Once the record is saved, it is easily verifiable, stays in the system, and is easily accessible.

By using the RecordsKeeper blockchain hospitals can store a patient's data, and it will remain immutable. Doctors can completely track a patient's progress. Also, there is another significant advantage: if a New Yorker has a medical emergency on vacation in Switzerland, for example, then the Swiss doctors can track her data from a New York hospital and can provide better treatment because the data is immutable and present on the distributed public ledger. So, we believe that with the help of RecordsKeeper, we can completely revolutionize the health industry.

3.7 SDKs for Recordskeeper Blockchain

Recordskeeper offers a number of SDKs to support various programming languages. Following are the SDKs available for RecordsKeeper Blockchain with their current status:

RecordsKeeper SDKs	Status
Recordskeeper Python SDK	Live
Node SDK	In Progress
Ruby SDK	In Progress
Android SDK	In Progress
Java SDK	In Progress
PHP SDK	In Progress
iOS SDK	Proposed

3.7.1 RecordsKeeper SDKs

RecordsKeeper Python SDK

Python libraries for recordskeeper. The RecordsKeeper Python SDK provides access to the python libraries of RecordsKeeper.

Installation

Install from PyPI using pip, a package manager for Python. There are two different packages for two different versions of python, for python2 version

```
pip install recordskeeper_python_lib        #for version2.0
pip install recordskeeper_python_lib3       #for version3.0 and greater
```

Compatibility

The python SDKs have been tested with python version 2.0 and python version 3.0 or greater.

Requirement

You have to create your own configuration file to interact with the blockchain with the required parameters given in the sample config file of these libraries.

Contents

genIndex, modindex, search

RecordsKeeper Python v2 SDK

Python libraries for recordskeeper. The RecordsKeeper Python SDK provides access to the python libraries of RecordsKeeper.

Installation

Install from PyPI using pip, a package manager for Python. There are two different packages for two different versions of python, for python2 version

```
pip install recordskeeper_python_lib
```

Compatibility

The SDK has been tested on python version 2.7.

Requirement

You have to create your own configuration file to interact with the blockchain with the required parameters given in the sample config file of these libraries.

Classes

genIndex, modindex, search

Address Class Usage

Library to work with RecordsKeeper addresses.

You can generate new addresses, check all addresses, check validity of an address, check mining permission of an address, check balance of an address and import address on the node by using Address class. You just have to pass required parameters to invoke the pre-defined functions.

Libraries

Import these python libraries first to get started with the functionality.

```
import requests
import json
from requests.auth import HTTPBasicAuth
import yaml
import sys
import binascii
```

Creating Connection

Entry point for accessing Address class resources.

- URL: Url to connect to the chain ([RPC Host]:[RPC Port])
- Chain-name: chain name

```
with open("config.yaml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile)
```

```
network = cfg['network']  #network variable to store the network that you want to␣
↪access
```

```
url = network['url']
chain = network['chain']
```

Node Authentication

Importing values from config file.

- User name: The rpc user is used to call the APIs.

- Password: The rpc password is used to authenticate the APIs.

```
user = network['rkuser']
password = network['passwd']
```

Now we have node authentication credentials.

Address Class

class Address

Address class is used to call address related functions like generate new addresses, list all the node addresses, check if the given address is valid or not, check if given address has mining permission or not, check balance of an address and import address on the node.

1. Generate new address on the node's wallet

getAddress() function is used to generate a new wallet address.

```
getAddress()

newAddress = getAddress()          #getAddress() function call

print newAddress                  # prints a new address
```

It will return a new address of the wallet.

2. Generate a new multisignature address

You have to pass these two arguments to the getMultisigAddress function call:

- nrequired: to pass the no of signatures that are must to sign a transaction
- key: pass a single variable of comma-seperated public keys

getMultisigAddress() function is used to generate a new multisignature address.

```
getMultisigAddress(nrequired, key)

newAddress = getMultisigAddress(nrequired, key)          #getMultisigAddress()␣
↪function call

print newAddress                            # prints a new address
```

It will return a new multisignature address on RecordsKeeper Blockchain.

3. Generate a new multisignature address on the node's wallet

You have to pass these two arguments to the getMultisigWalletAddress function call:

- nrequired: to pass the no of signatures that are must to sign a transaction
- key: pass a single variable of comma-seperated public keys

getMultisigWalletAddress() function is used to generate a new wallet address.

```
getMultisigWalletAddress(nrequired, key)

newAddress = getMultisigWalletAddress(nrequired, key)    #getMultisigWalletAddress()␣
↪function call

print newAddress                            #prints a new address
```

It will return a new multisignature address on the wallet.

4. List all addresses and no of addresses on the node's wallet

retrieveAddresses() function is used to list all addresses and no of addresses on the node's wallet.

```
retrieveAddresses()
result = retrieveAddresses()       #retrieveAddresses() function call

print result['address']             #prints all the addresses of the wallet
print result['address count']       #prints the address count
```

It will return all the addresses and the count of the addresses on the wallet.

5. Check validity of the address

You have to pass an address as the parameter to the checkifValid function call:

- Address: to check the validity

checkifValid() function is used to check validity of a particular address.

```
checkifValid()
addressCheck = checkifValid(address)  #checkifValid() function call

print addressCheck       # prints validity of the address
```

It will return if an address is valid or not.

6. Check if given address has mining permission or not

You have to pass an address as the parameter to the checkifMineAllowed function call:

- Address: to check the permission status

checkifMineAllowed() function is used to sign raw transaction by passing transaction hex of the raw transaction and the private key to sign the raw transaction.

```
checkifMineAllowed(address)
permissionCheck = checkifMineAllowed(address)   #checkifMineAllowed() function call

print permissionCheck       # prints permission status of the given address
```

It will return if mining permission is allowed or not.

7. Check address balance on a particular node

You have to pass an address as the parameter to the checkBalance function call:

- Address: to check the balance

checkBalance() function is used to check the balance of the address.

```
checkBalance(address)
address_balance = checkBalance(address)      #checkBalance() function call

print address_balance    # prints balance of the address
```

It will return the balance of the address on RecordsKeeper Blockchain.

8. Import a non-wallet address on RecordsKeeeper Blockchain

You have to pass an address as the parameter to the importAddress function call:

- Address: non-wallet address to import on a particular node importAddress() function is used to check the balance of the address.

```
importAddress(public_address)
response = importAddress(public_address)      #importAddress() function call

print response    # prints response whether address is successfully imported or not
```

It will return the response of the importAddress() function call.

Assets Class Usage

Library to work with RecordsKeeper assets.

You can create new assets, send assets and list all assets by using Assets class. You just have to pass the required parameters to invoke the pre-defined functions of Assets class.

Libraries

Import these python libraries first to get started with the functionality.

```
import requests
import json
from requests.auth import HTTPBasicAuth
import yaml
import sys
import binascii
```

Creating Connection

Entry point for accessing Address class resources.

- URL: Url to connect to the chain ([RPC Host]:[RPC Port])
- Chain-name: chain name

```
with open("config.yaml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile)
```

```
network = cfg['network']   #network variable to store the network that you want to␣
↪access
```

```
url = network['url']
chain = network['chain']
```

Node Authentication

Importing values from config file.

- User name: The rpc user is used to call the APIs.
- Password: The rpc password is used to authenticate the APIs.

```
user = network['rkuser']
password = network['passwd']
```

Now we have node authentication credentials.

Assets Class

class Assets

Assets class is used to call asset related functions like create assets, send assets and list assets functions which are used on the RecordsKeeeper Blockchain.

1. Create Assets on the RecordsKeeper Blockchain

createAsset() function is used to create or issue an asset.

```
createAsset(address, asset_name, asset_qty)

txid = createAsset(address, asset_name, asset_qty)          #createAsset() function␣
↪call

print txid                    # prints transaction id of the issued asset
```

It will return the transaction id of the issued asset.

2. Send Assets to a particular address on the RecordsKeeper Blockchain

You have to pass these three arguments to the sendAsset function call:

- address: address which will send the asset
- asset_name: name of the asset
- qty: quantity of asset to be sent

sendAsset() function is used to send an asset.

```
sendAsset(address, assetname, qty)

txid = sendAsset(address, assetname, qty)   #sendAsset() function call

print txid  #prints transaction id of the sent asset
```

It will return the transaction id of the sent asset.

3. List all assets on the RecordsKeeper Blockchain

retrieveAssets() function is used to list all the assets, no of the assets, issued quantity of the assets and issued transaction id of all the assets on the RecordsKeeper Blockchain.

```
retrieveAssets()
result = retrieveAssets()       #retrieveAssets() function call

print result['name']            #prints name of all the assets
print result['asset count']     #prints total asset count
print result['id']              #prints assets issued quantity
print result['qty']             #prints assets issued transaction id
```

It will return all the assets, the count of the assets, issued quantity of the assets and issued transaction id of the assets on the RecordsKeeper Blockchain.

Block Class Usage

Library to work with RecordsKeeper block informaion.

You can collect block information like block's transaction count, blocktime, blockhash, miner of the block by using block class.You just have to pass the required parameters to invoke the pre-defined functions of Block class.

Libraries

Import these python libraries first to get started with the functionality.

```
import requests
import json
from requests.auth import HTTPBasicAuth
import yaml
import sys
```

Creating Connection

Entry point for accessing Address class resources.

- URL: Url to connect to the chain ([RPC Host]:[RPC Port])
- Chain-name: chain name

```
with open("config.yaml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile)
```

```
network = cfg['network']  #network variable to store the network that you want to␣
↪access
```

```
url = network['url']
chain = network['chain']
```

Node Authentication

Importing values from config file.

- User name: The rpc user is used to call the APIs.
- Password: The rpc password is used to authenticate the APIs.

Default value of network is **Test-net**, you can change its value to select mainnet or testnet

```
user = network['rkuser']
password = network['passwd']
```

Now we have node authentication credentials.

Block Class

class Block

Block class is used to call block related function like blockinfo or retrieveBlocks which are used to retrieve block details like block's hash value, size, nonce, transaction ids, transaction count, miner address, previous block hash, next block hash, merkleroot, blocktime and difficulty of the queried block.

1. Block info to retrieve block information

You have to pass block height as the argument to the blockinfo function call:

- Block height: height of the block of which you want to collect information

```
blockinfo(block_height)
result = blockinfo(block_height)

print result['txcount']      #prints transaction count of the block
print result['tx']           #prints transaction ids of the block
print result['size']         #prints size of the block
print result['blockhash']    #prints hash value of the block
print result['nonce']        #prints nonce of the block
print result['miner']        #prints miner's address of the block
print result['nextblock']    #prints next block's hash
print result['prevblock']    #prints previous block's hash
print result['merkleroot']   #prints merkle root of the block
print result['blocktime']    #prints time at which block is mined
print result['difficulty']   #prints difficulty of the block
```

It will return transaction ids, transaction count, nonce, size, hash value, previous block's hash value, next block hash value, merkle root, difficulty, blocktime and miner address of the block.

2. Retrieve a range of blocks on RecordsKeeper chain

You have to pass range of blocks i.e. blockrange can be 10-15. It can be passed as the argument to the retrieveBlocks function call:

- Block range: range of the block of which you want to collect info

. code-block:: python

> . code-block:: python
>
> retrieveBlocks(block_range) result = retrieveBlocks(block_range)
>
> print result['blockhash'] #prints hash of the blocks print result['miner'] #prints miner of the blocks print result['blocktime'] #prints block time of the blocks print result['tx count'] #prints transaction count of the blocks

It will return blockhash, miner address, blocktime and transaction count of the queried blocks.

Blockchain Class Usage

Library to work with Blockchain class in RecordsKeeper Blockchain.

You can get chain information, node information, node's permissions, pending transaction information and node balance by using Blockchain class. You just have to pass the required parameters to invoke the pre-defined functions of Blockchain class.

Libraries

Import these python libraries first to get started with the functionality.

```
import requests
import json
from requests.auth import HTTPBasicAuth
import yaml
import sys
import binascii
```

Creating Connection

Entry point for accessing Address class resources.

- URL: Url to connect to the chain ([RPC Host]:[RPC Port])
- Chain-name: chain name

```
with open("config.yaml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile)
```

```
network = cfg['network']  #network variable to store the network that you want to␣
↪access
```

```
url = network['url']
chain = network['chain']
```

Node Authentication

Importing values from config file.

- User name: The rpc user is used to call the APIs.
- Password: The rpc password is used to authenticate the APIs.

```
user = network['rkuser']
password = network['passwd']
```

Now we have node authentication credentials.

Blockchain Class

class Blockchain

Blockchain class is used to call blockchain related functions like retrieving blockchain parameters, retrieving node's information, retrieving mempool's information, retrieving node's permissions and check node's balance functions which are used on the RecordsKeeper Blockchain.

1. Retrieve Blockchain parameters of RecordsKeeper Blockchain

getChainInfo() function is used to retrieve Blockchain parameters.

```
getChainInfo()

result = getChainInfo()                 #getChainInfo() function call

print result['chain-protocol']          #prints blockchain's protocol
```

(continues on next page)

(continued from previous page)

```
print result['chain-description']       #prints blockchain's description
print result['root-stream-name']        #prints blockchain's root stream
print result['maximum-blocksize']       #prints blockchain's maximum block size
print result['default-network-port']    #prints blockchain's default network port
print result['default-rpc-port']        #prints blockchain's default rpc port
print result['mining-diversity']        #prints blockchain's mining diversity
print result['chain-name']              #prints blockchain's name
```

It will return the information about RecordsKeeper blockchain's parameters.

2. Retrieve node's information on RecordsKeeper Blockchain

getNodeInfo() function is used to retrieve node's information on RecordsKeeper Blockchain.

```
getNodeInfo()
result = getNodeInfo()       #getNodeInfo() function call

print result['node balance']      #prints balance of the node
print result['synced blocks']      #prints no of synced blocks
print result['node address']      #prints node's address
print result['difficulty']      #prints node's difficulty
```

It will return node's balance, no of synced blocks, node's address and node's difficulty.

3. Retrieve permissions given to the node on RecordsKeeper Blockchain

permissions() function is used to retrieve node's permissions.

```
permissions()
allowed_permissions = permissions()                #permissions() function call

print allowed_permissions      # prints permissions available to the node
```

It will return the permissions available to the node.

4. Retrieve pending transaction's information on RecordsKeeper Blockchain

getpendingTransactions() function is used to retrieve pending transaction's information like no of pending transactions and the pending transactions.

```
getpendingTransactions()
result = getpendingTransactions(address)    #getpendingTransactions() function call

print result['tx']              #prints pending transactions
print result['tx_count']        #prints pending transaction count
```

It will return the information of pending transactions on Recordskeeper Blockchain.

5. Check node's total balance

checkNodeBalance() function is used to check the total balance of the node.

```
checkNodeBalance()
node_balance = checkNodeBalance()   #checkNodeBalance() function call

print node_balance   #prints total balance of the node
```

It will return the total balance of the node on the RecordsKeeper Blockchain.

Permissions Class Usage

Library to work with Permission class in RecordsKeeper Blockchain.

You can grant or revoke permissions like create, send, recieve, mine, admin, connect, issue and activate by using Permissions class. You just have to pass the required parameters to invoke the pre-defined functions.

Libraries

Import these python libraries first to get started with the functionality.

```
import requests
import json
from requests.auth import HTTPBasicAuth
import yaml
import sys
import binascii
```

Creating Connection

Entry point for accessing Address class resources.

- URL: Url to connect to the chain ([RPC Host]:[RPC Port])
- Chain-name: chain name

```
with open("config.yaml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile)
```

```
network = cfg['network']  #network variable to store the network that you want to␣
↪access
```

```
url = network['url']
chain = network['chain']
```

Node Authentication

Importing values from config file.

- User name: The rpc user is used to call the APIs.
- Password: The rpc password is used to authenticate the APIs.

```
user = network['rkuser']
password = network['passwd']
```

Now we have node authentication credentials.

Permissions Class

class Permissions

Permissions class is used to call permissions related functions like grant and revoke permissions of an address on the RecordsKeeper Blockchain.

1. Grant Permissions to an address on the RecordsKeeper Blockchain

You have to pass these two arguments to the grantPermission function call:

- Permissions: list of comma-seperated permissions passed as a string
- Address: to which you have to grant permission

grantPermission() function is used to grant permissions like connect, send, receive, create, issue, mine, activate, admin to an address on RecordsKeeper Blockchain.

```
grantPermission(address, permissions)

result = grantPermission(address, permissions)   #grantPermission() function call

print txid  #prints response of the grant permision transaction
```

It will return the transaction id of the permission transaction.

2. Revoke Permissions to an address on the RecordsKeeper Blockchain

You have to pass these two arguments to the revokePermission function call:

- Permissions: list of comma-seperated permissions passed as a string
- Address: to which you have to grant permission

revokePermission() function is used to revoke permissions like connect, send, receive, create, issue, mine, activate, admin to an address on RecordsKeeper Blockchain.

```
revokePermission(address, permissions)
result = revokePermission(address, permissions)   #revokePermission() function call

print result   #prints response of the revoke permision transaction
```

It will return the transaction id of the permission transaction.

Stream Class Usage

Library to work with RecordsKeeper streams.

You can publish, retrieve and verify stream data by using Stream class. You just have to pass the required parameters to invoke the pre-defined functions of the Stream class.

Libraries

Import these python libraries first to get started with the functionality.

```
import requests
import json
from requests.auth import HTTPBasicAuth
import yaml
import binascii
import sys
```

Creating Connection

Entry point for accessing Address class resources.

- URL: Url to connect to the chain ([RPC Host]:[RPC Port])
- Chain-name: chain name

```
with open("config.yaml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile)
```

```
network = cfg['network']  #network variable to store the network that you want to␣
↪access
```

```
url = network['url']
chain = network['chain']
```

Node Authentication

Importing values from config file.

- User name: The rpc user is used to call the APIs.
- Password: The rpc password is used to authenticate the APIs.

```
user = network['rkuser']
password = network['passwd']
```

Now we have node authentication credentials.

Stream Class

class Stream

Stream class to call stream related functions like publish, retrieve, retrieveWithAddress, retrieveWithKey and verify data functions which are used to publish data into the stream, retrieve data from the stream and verify data from the stream.

1. Publish

You have to pass these four parameters to the publish function call:

- Data Hex of the data to be published
- Address of the publihser
- Stream to which you want your data to be published
- key Value for the data to be published

```
publish(address, stream, key, data)

txid = publish(address, stream, key, data)

print txid    #prints the transaction id of the data published
```

It will return the transaction id of the published data.

2. Retrieve an existing item from a particular stream against a transaction id

You have to pass these two arguments to the retrieve function call:

- Stream name: which you want to access
- Transaction id: id of the data that you want to retrieve

```
retrieve(stream, txid)     #call retrieve function with stream and txid as the␣
↪required parameters
result = retrieve(stream, txid)

print result  #prints data
```

It will return the data corresponding to the passed transaction id.

3. Retrieve an item against a particular publisher address

You have to pass these three arguments to the retrieveWithAddress function call:

- Stream name: which you want to access
- Publisher address: address of the data publisher you want to verify
- Count: no of items you want to retrieve

```
retrieveWithAddress(stream, address, count)
result = retrieveWithAddress(stream, address, count)

print result['key']       #prints key value of the data
print result['txid']      #prints transaction id of the data
print result['data']      #prints raw data
```

It will return the key value, raw data and transaction id of the published item.

4. Retrieve an item against a particular key value

You have to pass these three arguments to the retrieveWithKey function call:

- Stream name: which you want to access
- Key: key value of the published data you want to verify
- Count: no of items you want to retrieve

```
retrieveWithKey(stream, key, count)
result = retrieveWithKey(stream, key, count)

print result['publisher']    #prints publisher's address of the published data
print result['txid']         #prints transaction id of the data
print result['data']         #prints raw data
```

It will return the key value, raw data and transaction id of the published item.

5. Verify an data item on a particular stream of RecordsKeeper Blockchain

You have to pass these three arguments to the retrieveWithKey function call:

- Stream name: which you want to access
- Data: against which you want to make a query
- Count: count of items which will be queried

```
verifyData(stream, data, count)
result = verifyData(stream, data, count)

print result    #prints if verification is successful or not
```

It will return the result if verification is successful or not.

6. Retrieve data items on a particular stream of RecordsKeeper Blockchain

You have to pass these two arguments to the verifyWithKey function call:

- Stream name: which you want to access
- Count: count of items which will be queried

```
retrieveItems(stream, count)
result = retrieveItems(stream, count)

print result['address']    #prints address of the publisher of the item
print result['key']        #prints key value of the stream itme
print result['data']       #prints raw data published
print result['txid']       #prints transaction id of the item published
```

It will return the address, key value, data and transaction id of the stream item published.

Transaction Class Usage

Library to work with RecordsKeeper transactions.

You can send transaction, create raw transaction, sign raw transaction, send raw transaction, send signed transaction, retrieve transaction information and calculate transaction's fees by using transaction class. You just have to pass the required parameters to invoke the pre-defined functions.

Libraries

Import these python libraries first to get started with the functionality.

```
import requests
import json
from requests.auth import HTTPBasicAuth
import yaml
import sys
import binascii
```

Creating Connection

Entry point for accessing Address class resources.

- URL: Url to connect to the chain ([RPC Host]:[RPC Port])
- Chain-name: chain name

```
with open("config.yaml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile)
```

```
network = cfg['network']  #network variable to store the network that you want to␣
↪access
```

```
url = network['url']
chain = network['chain']
```

Node Authentication

Importing values from config file.

- User name: The rpc user is used to call the APIs.
- Password: The rpc password is used to authenticate the APIs.

```
user = network['rkuser']
password = network['passwd']
```

Now we have node authentication credentials.

Transaction Class

class Transaction

Transaction class is used to call transaction related functions like create raw transaction, sign transaction, send transaction, retrieve transaction and verify transaction functions which are used to create raw transactions, send transactions, sign transactions, retrieve transactions and verify transactions on the RecordsKeeeper Blockchain.

1. Send Transaction without signing with private key

You have to pass these three arguments to the sendTransaction function call:

- Transaction's sender address
- Transaction's reciever address
- Amount to be sent in the transaction

sendTransaction() function is used to send transaction by passing reciever's address, sender's address and amount.

```
sendTransaction(sender_address, reciever_address, data, amount)

txid = sendTransaction(sender_address, reciever_address, data, amount)

print txid   #prints transaction id of the sent transaction
```

It will return the transaction id of the raw transaction.

2. Send Transaction by signing it with private key

You have to pass these four arguments to the sendSignedTransaction function call:

- Transaction's sender address
- Transaction's reciever address
- Amount to be sent in transaction
- Private key of the sender's address

sendSignedTransaction() function is used to send transaction by passing reciever's address, sender's address, private key of the sender and the amount. In this function private key is required to sign transaction.

```
sendSignedTransaction(sender_address, reciever_address, amount, private_key, data)
transaction_id = sendSignedTransaction(sender_address, reciever_address, amount,␣
↪private_key, data)

print transaction_id        #prints transaction id of the signed transaction
```

It will return transaction id of the signed transaction.

3. Create raw transaction

You have to pass these three arguments to the createRawTransaction function call:

- Transaction's sender address
- Transaction's reciever address
- Amount to be sent in transaction

createRawTransaction() function is used to create raw transaction by passing reciever's address, sender's address and amount.

```
createRawTransaction(sender_address, reciever_address, amount, data)
tx_hex = createRawTransaction(sender_address, reciever_address, amount, data)

print tx_hex       #prints transaction hex of the raw transaction
```

It will return transaction hex of the raw transaction.

4. Sign raw transaction

You have to pass these three arguments to the signRawTransaction function call:

- Transaction hex of the raw transaction
- Private key to sign raw transaction

signRawTransaction() function is used to sign raw transaction by passing transaction hex of the raw transaction and the private key to sign the raw transaction.

```
signRawTransaction(tx_hex, private_key)
signed_hex = signRawTransaction(txHex, private_key)

print signed_hex      #prints signed transaction hex of the raw transaction
```

It will return signed transaction hex of the raw transaction.

5. Send raw transaction

You have to pass these three arguments to the sendRawTransaction function call:

- Signed transaction hex of the raw transaction

sendRawTransaction() function is used to send raw transaction by passing signed transaction hex of the raw transaction.

```
sendRawTransaction(signed_txHex)
tx_id = sendRawTransaction(signed_txHex)

print tx_id     #prints transaction id of the raw transaction
```

It will return transaction id of the raw transaction sent on to the Blockchain.

6. Retrieve a transaction from the Blockchain

You have to pass given argument to the retrieveTransaction function call:

- Transaction id of the transaction you want to retrieve

retrieveTransaction() function is used to retrieve transaction's information by passing transaction id to the function.

```
retrieveTransaction(tx_id)
result = retrieveTransaction(tx_id)

print result['sent data']       #prints sent data
print result['sent amount']     #prints sent amount
```

It will return the sent data and sent amount of the retrieved transaction.

7. Calculate a particular transaction's fee on RecordsKeeper Blockchain

You have to pass these two arguments to the getFee function call:

- Transaction id of the transaction you want to calculate fee for
- Sender's address

getFee() function is used to calculate transaction's fee by passing transaction id and sender's address to the function.

```
getFee(address, tx_id)
Fees = getFee(address, tx_id)

print (Fees)  #prints fees consumed in the verified transaction
```

It will return the fees consumed in the transaction.

Wallet Class Usage

Library to work with RecordsKeeper wallet class.

You can create wallet, dump wallet into a file, backup wallet into a file, import wallet from a file, lock wallet, unlock wallet, change wallet's password, retrieve private key, retrieve wallet's information, sign and verify message by using wallet class. You just have to pass parameters to invoke the pre-defined functions.

Libraries

Import these python libraries first to get started with the functionality.

```
import requests
import json
from requests.auth import HTTPBasicAuth
import yaml
import sys
import binascii
```

Creating Connection

Entry point for accessing Address class resources.

- URL: Url to connect to the chain ([RPC Host]:[RPC Port])
- Chain-name: chain name

```
with open("config.yaml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile)
```

```
network = cfg['network']  #network variable to store the network that you want to␣
↪access
```

```
url = network['url']
chain = network['chain']
```

Node Authentication

Importing values from config file.

- User name: The rpc user is used to call the APIs.
- Password: The rpc password is used to authenticate the APIs.

```
user = network['rkuser']
password = network['passwd']
```

Now we have node authentication credentials.

Wallet Class

class Wallet

Wallet class is used to call wallet related functions like create wallet, retrieve private key of wallet address, retrieve wallet's information, dump wallet, lock wallet, unlock wallet, change wallet's password, create wallet's backup, import wallet's backup, sign message and verify message functions on RecordsKeeeper Blockchain.

1. Create wallet on RecordsKeeper blockchain

createWallet() function is used to create wallet on RecordsKeeper blockchain

```
createWallet()

result = createWallet()

print result['public address']        #prints public address of the wallet
print result['private key']           #prints private key of the wallet
print result['public key']            #prints public key of the wallet
```

It will return the public address, public key and private key.

2. Retrieve private key of an address

You have to pass an address as the parameter to the getPrivateKey function call:

- Public Address: address whose private key is to be retrieved

getPrivateKey() function is used to retrieve private key of the given address.

```
getPrivateKey(public_address)
privkey = getPrivateKey(public_address)

print privkey         #prints private key of the given address
```

It will return private key of the given address.

3. Retrieve node wallet's information

retrieveWalletinfo() function is used to retrieve node wallet's information.

```
retrieveWalletinfo()
result = retrieveWalletinfo()

print result['balance']       #prints wallet's balance
print result['tx count']      #prints wallet transaction count
print result['unspent tx']    #prints unspent wallet transactions
```

It will return the wallet's balance, transaction count and unspent transactions.

4. Create wallet's backup

You have to pass these three arguments to the backupWallet function call:

- Filename: wallet's backup file name

backupWallet() function is used to create backup of the wallet.dat file.

```
backupWallet(filename)
result = backupWallet(filename)

print result       #prints result
```

It will return the response of the backup wallet function. The backup of the wallet is created in your chain's directory and you can simply access your file by using same filename that you have passed with the backupwallet function. Creates a backup of the wallet.dat file in which the node's private keys and watch-only addresses are stored. The backup is created in file filename. Use with caution – any node with access to this file can perform any action restricted to this node's addresses.

5. Import backup wallet

You have to pass these three arguments to the importWallet function call:

- Filename: wallet's backup file name

importWallet() function is used to import wallet's backup file.

```
importWallet(filename)
result = importWallet(filename)

print result     #prints result
```

It will return the response of the import wallet function. It will import the entire set of private keys which were dumped (using dumpwallet) into file filename.

6. Dump wallet on RecordsKeeper blockchain

You have to pass these three arguments to the dumpWallet function call:

- Filename: file name to dump wallet in

dumpWallet() function is used to retrieve transaction's information by passing transaction id to the function.

```
dumpWallet(filename)
result = dumpWallet(filename)

print (result)    #prints result
```

It will return the response of the dump wallet function. Dumps the entire set of private keys in the wallet into a human-readable text format in file filename. Use with caution – any node with access to this file can perform any action restricted to this node's addresses.

7. Locking wallet with a password on RecordsKeeper Blockchain

You have to pass password as an argument to the lockWallet function call:

- Password: password to lock the wallet

lockWallet() function is used to verify transaction's information by passing transaction id and sender's address to the function.

```
lockWallet(password)
result = lockWallet(password)

print (result)     #prints result
```

It will return the the response of the lock wallet function. This encrypts the node's wallet for the first time, using passphrase as the password for unlocking. Once encryption is complete, the wallet's private keys can no longer be retrieved directly from the wallet.dat file on disk, and chain will stop and need to be restarted. Use with caution – once a wallet has been encrypted it cannot be permanently unencrypted, and must be unlocked for signing transactions with the unlockwallet function.

8. Unlocking wallet with the password on RecordsKeeper Blockchain

You have to pass these two arguments to the unlockWallet function call:

- Password: password to unlock the wallet
- unlocktime: seconds for which wallet remains unlock

unlockWallet() function is used to verify transaction's information by passing transaction id and sender's address to the function.

```
unlockWallet(password, unlock_time)
result = unlockWallet(password, unlock_time)

print (result)      #prints result
```

It will return the response of the unlock wallet function. This uses passphrase to unlock the node's wallet for signing transactions for the next timeout seconds. This will also need to be called before the node can connect to other nodes or sign blocks that it has mined.

9. Change wallet's password

You have to pass these two arguments to the changeWalletPassword function call:

- Old Password: old password of the wallet
- New Password: new password of the wallet

changeWalletPassword() function is used to change wallet's password and set new password.

```
changeWalletPassword(old_password, new_password)
result = changeWalletPassword(password, new_password)
```

(continues on next page)

(continued from previous page)

```
print (result)      #prints result
```

This changes the wallet's password from old-password to new-password.

10. Sign Message on RecordsKeeper Blockchain

You have to pass these two arguments to the signMessage function call:

- Message: message to send
- Private Key: private key of the sender's wallet address

signMessage() function is used to change wallet's password and set new password.

```
signMessage(private_key, message)
signedMessage = signMessage(priavte_key, message)

print (signedMessage)  #prints signed message
```

It will return the signed message.

11. Verify Message on RecordsKeeper Blockchain

You have to pass these three arguments to the verifyMessage function call:

- Message: message to send
- Private Key: private key of the sender's wallet address

verifyMessage() function is used to change wallet's password and set new password.

```
verifyMessage(address, signedMessage, message)
validity = verifyMessage(address, signedMessage, message)

print (validity)   #prints validity of the message
```

It will return the validity of the message.

RecordsKeeper Python v3 SDK

Python libraries for recordskeeper. The RecordsKeeper Python SDK provides access to the python libraries of RecordsKeeper.

Installation

Install from PyPI using pip, a package manager for Python. There are two different packages for two different versions of python, for python2 version

```
pip install recordskeeper_python_lib3
```

Compatibility

The SDK has been tested with Python 3.5.

Requirement

You have to create your own configuration file to interact with the blockchain with the required parameters given in the sample config file of these libraries.

Classes

genIndex, modindex, search

3.8 Contributing

We encourage open source contributions!

To get started, you can try building from the source code in order to familiarize yourself with the components of RecordsKeeper and the build process. Also, it may be useful to become well-versed with command line functions for RecordsKeeper.

In particular, we need help in the following areas:

> Improving the documentation Responding to questions from other users on StackExchange and the RecordsKeeper Github Fixing and responding to RecordsKeeper GitHub issues, especially those tagged as up-for-grabs which are meant as introductory issues for external contributors.

3.8.1 How to Report Issues

To report an issue, please use the GitHub issues tracker. When reporting issues, please mention the following details:

- Which version of RecordsKeeper you are using
- What the source code was (if applicable)
- Which platform you are operating on
- How to reproduce the issue
- What the result of the issue was
- What the expected behavior is

Reducing the source code that caused the issue to a bare minimum is always very helpful and sometimes even clarifies a misunderstanding.

3.8.2 Workflow for Pull Requests

In order to contribute, please fork off of the develop branch and make your changes there. Your commit messages should detail why you made your change, in addition to what you did and whether it may affect the other parts of RecordsKeeper.

If you need to pull in any changes from develop after making your fork (for example, to resolve potential merge conflicts), please avoid using git merge and instead git rebase your branch.

Additionally, if you are writing a new feature, please ensure you write appropriate unit test cases and place them under test/.

However, if you are making a larger change, please consult with the RecordsKeeper Development Team.

Finally, please make sure you respect the coding standards for this project. Also, even though we do CI testing, please test your code and ensure that it builds locally before submitting a pull request.

Thank you for your help!

3.9 Frequently Asked Questions

3.9.1 RecordsKeeper FAQ

What is RecordsKeeper?

RecordsKeeper is a public, mineable blockchain for record keeping & data security. It can also be seen as an open-source platform for private blockchains, which offers a rich set of features including extensive configurability, rapid deployment, permissions management, native assets, and data streams.

You can publish your data in key-value pair format on the RecordsKeeper blockchain.

Is RecordsKeeper a storage, database, or blockchain?

RecordsKeeper is a key-value pair based database running on blockchain. It stores every record as a key-value pair as a part of XRK transactions. Alternatively, you can regard it as an amalgamation of all three (storage, database and blockchain) as it offers storage as a database on the public blockchain.

How is RecordsKeeper different from traditional storage systems?

Since RecordsKeeper is a public blockchain, it consists of several decentralized nodes (aka servers) which verify each and every transaction while adding new blocks into the blockchain. Each node participates in administration: all nodes verify new additions to the blockchain, and are capable of entering new data into RecordsKeeper. Traditional databases allow the modification of data, but with the RecordsKeeper blockchain the data is secured and cannot be tampered with, altered, or modified once published and confirmed in the RecordsKeeper blockchain ledger.

How is RecordsKeeper different from Storj, Filecoin, and Siacoin?

RecordsKeeper is based on a simple mechanism to utilize transaction size to store data as compared to other 3rd parties which uses the rental of other user's hard drives to create a network of storage. RecordsKeeper blockchain is based on key-value pairs, which reduces the need to source storage space from other parties.

How is RecordsKeeper different from Blockchain DB?

RecordsKeeper offers ease of access to the database. You can use the JSON RPC commands to interact with the blockchain, which keeps RecordsKeeper in line with the Bitcoin blockchain.

How does RecordsKeeper work?

RecordsKeeper allows the user to upload data (in key-value pair format) onto the blockchain with ease and in a single XRK transaction. Users can upload multiple entries in the form of key-value pairs, but it has to be one pair per transaction. RecordsKeeper is a public blockchain, and it also allows users to retrieve their uploaded data by the use of a key. The cost of data upload is calculated as per the data size and awarded to the miners in the form of XRK.

What kind of data can I publish on RecordsKeeper?

RecordsKeeper allows different data formats, ranging from JSON, XML, Hex, or Objects, to Simple text. Users are allowed to upload data and retrieve data up to the maximum size of 2 MB per transaction. Please remember that the fees in XRK are determined by how much data you upload.

What's the maximum amount of data I can publish on RecordsKeeper?

RecordsKeeper allows OP_RETURN transactions of XRK up to 2 MB per transaction and other transactions up to 4 MB. If you are using the key-value database in RecordsKeeper blockchain, then the maximum size allowed is 2 MB per transaction.

Can I upload an entire file on RecordsKeeper?

Technically, it is possible but it is not recommended. A file can have a large amount of raw data which does not necessarily need to be kept secure and may cost you a lot in terms of XRK fees. It is highly recommended to upload the textual juice of the file or record in the structures textual data like JSON/XML. In any case, the maximum size of published records can be up to 2 MB per XRK transaction. Alternatively, if you wish to upload larger files upto 5 GB in size, then you can upload the hash of the file over the RecordsKeeper blockchain to publish the proof-of-existence, proof-of-integrity tec. or contact us for further assistance.

What are the blockchain specifications for RecordsKeeper?

The RecordsKeeper specifications are as follows:

- Avg. Block Size: 250 bytes
- Max Block Size: 8 MB
- Avg. Block Frequency: 15 seconds
- Mining Algorithm: PoW
- Premined Tokens: 300 million XRK
- Mining Reward: 10 XRK
- Max Transaction Size: 4 MB
- Max OP_RETURN Transaction Size: 2 MB

What is the maximum block size for the RecordsKeeper blockchain?

The maximum block size on the RecordsKeeper blockchain is 8 MB (8388608 bytes).

How many confirmed tx/sec can RecordsKeeper achieve?

The average size of transactions on the RecordsKeeper blockchain is 250 bytes and the maximum size of a block is 8 MB (8388608 bytes). Thus, you can have up to 33554 transactions per block on average. The average block time is 15 seconds, so you can have up to 2236 transactions/sec if they are all small size transactions. If the transaction sizes are larger, the number of transactions decrease accordingly.

What are the use cases of RecordsKeeper?

There are several use cases for RecordsKeeper ranging from KYC verifications, supply chain management, manufacturing, health record management, academic certifications, and employment credentials verification. You can view all the use cases in the RecordsKeeper whitepaper .

Is data uploaded to RecordsKeeper encrypted by default?

No. The data uploaded to RecordsKeeper is not encrypted by default. If you wish to encrypt the data to make it available on blockchain while keeping it private, then you can always encrypt it in your application layer. This will ensure its immutability, as well as its privacy.

Who can retrieve published records?

Anyone who has the key or the transaction ID of the published records can retrieve the data. You can also retrieve records sent to a specific address on the RecordsKeeper blockchain using Blockchain Explorer or its native APIs.

How can I verify published records?

To verify published records, you can access the data using the key from the RecordsKeeper blockchain, and then compare it with your locally stored record. If both the records match exactly, then your record's integrity and immutability has been maintained in your local storage. If they do not, then it clearly means that someone has tampered with the local records.

What is the cost of publishing records?

The current fee for publishing records is 0.1 XRK/KB of data. This can vary due to supply and demand.

3.9.2 Mining FAQ

Can I mine XRK?

Yes. Anyone can become a miner with RecordsKeeper. You need to send us your mining address for permissions, and then you can start mining. You can follow the mining guide for further instructions here. (Link to mining guide).

How can I mine XRK?

Follow the mining guide instructions from (Link to mining guide) to start mining XRK.

What are the minimum hardware requirements for XRK mining?

Anyone with a personal laptop/computer can begin mining for XRK. The minimum system requirements are as follows:

- Linux: 64-bit, supports Ubuntu 12.04+, CentOS 6.2+, Debian 7+, Fedora 15+, RHEL 6.2+.
- Windows: 64-bit, supports Windows 7, 8, 10, Server 2008 or later.
- Mac: 64-bit, supports OS X 10.12 (we hope to support earlier versions soon).
- 512 MB of RAM

- 1 GB of disk space

3.9.3 XRK FAQ

What is XRK?

XRK is the name (or ticker) of the tokens which are used as an incentive and payment model for uploading records and data onto the RecordsKeeper blockchain. XRK tokens are used as fees for uploading the records over the RecordsKeeper blockchain.

What is the use of XRK?

XRK tokens are used to upload records over the RecordsKeeper blockchain. The Blockchain computes the required fees for the uploaded records and awards those fees to miners who confirm the transaction carrying the data.

What is the value of XRK?

The current value of XRK is 1 BTC (Bitcoin) = 20,000 XRK. This value is subject to change as per supply and demand.

How are XRK generated?

The premined XRK tokens are in total 300 million which you can buy and use for RecordsKeeper. You can also generate and earn more XRK through mining. Refer the mining guide to set up the mining for XRK (Link to mining guide)

Can XRK tokens be destroyed or burned?

XRK cannot be destroyed or burned. However, you can send XRK to a NOP_RETURN transaction thus making them 'un-spendable' for further transactions.

How many total XRK tokens are in circulation?

There are 300 million premined XRK tokens in total 300. Their value keeps increasing as more XRK tokens are added through mining rewards, which are 10 XRK per block.

How can I get Testnet XRK tokens?

Testnet XRK tokens are available for the community to build and deploy applications on the RecordsKeeper blockchain. You can get Testnet XRK through the RecordsKeeper faucet .

Do I need to buy XRK to use the Demo?

The RecordsKeeper Demo provides new users with 1 XRK coin over the mainnet, which can be used to publish transaction on the RecordsKeeper blockchain. The maximum amount of data which can be published is based upon the fees. If you want to publish large amounts of data, you need to buy more XRK tokens. To buy XRK, please contact us here.

Who gets the XRK which are spent as transaction fees?

The miner who confirms the transaction gets the XRK spent in transaction fees.

On which exchanges is XRK listed?

We have not been listed on any exchange as of now. However, we are in talk with multiple exchanges for this. Please subscribe to our newsletter to get updates on exchange listing.

How can I purchase XRK in bulk?

To buy XRK in bulk, you can contact us here.

3.10 Contact Us

3.10.1 Tell Us About Your Current Challenges

We thrive when coming up with innovative ideas but also understand that a smart concept should be supported with measurable results and individuals.

Join us at the following:

- Facebook
- Twitter
- Telegram

If you still have questions, you can try searching or asking on the RecordsKeeper FAQ, or send us you queries at RecordsKeeper Contact Us. Ideas for improving RecordsKeeper or this documentation are always welcome!

Index