index

July 12, 2021

[1]:
```
#hide
```

1 Sovryn Knowledge Graph

> A knowledge graph of the Sovryn protocol and all its transactions.

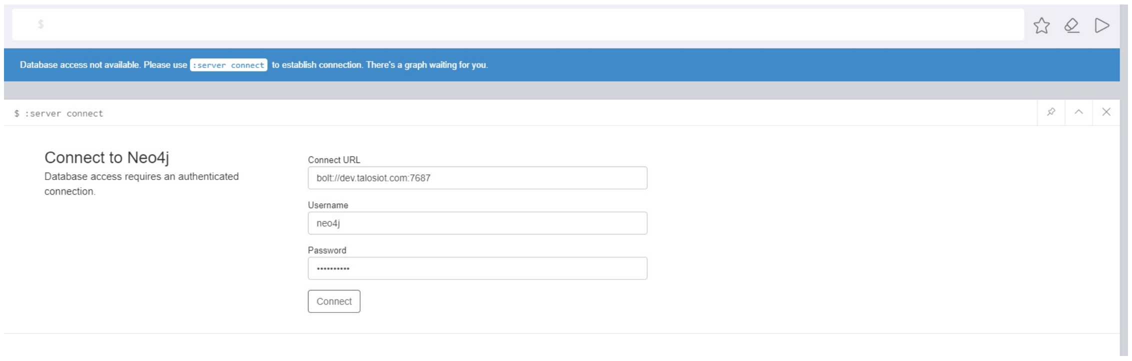
Explore the sovryn protocol, follow the money, or harvest any data related to Sovryn.

1.1 Installation

You'll need a working docker and docker-compose.

```
git clone https://github.com/except-pass/sovrynkg
docker-compose up -d
```

This starts a neo4j database. You can access it using a browser on port `7474`. The default user name is `neo4j` and the password is `staysovryn`. Point a browser to `http://localhost:7474` and login.



Database access not available. Please use `:server connect` to establish connection. There's a graph waiting for you.

```
$ :server connect
```

Connect to Neo4j

Database access requires an authenticated connection.

Connect URL: bolt://dev.talosiot.com:7687

Username: neo4j

Password: ••••••••••

1.2 Tour

You can query this database using a special query langauge called Cypher. A cypher tutorial is beyond the scope of this writeup, but we will go through some cool things you can do.

You can type any query directly into or into a `knowledge_graph.Query` object. First, lets see a few blocks with available data.

```
[2]: from sovrynkg.knowledge_graph import Query
     q = Query()
     q.add("MATCH (b:Block) RETURN b.height as height ORDER BY height LIMIT 10")
     q.data()
```

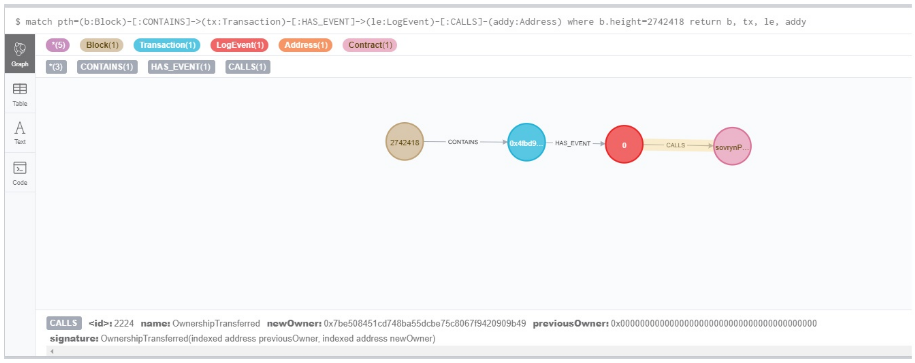
```
[2]: [{'height': 2742418},
      {'height': 2742441},
      {'height': 2742445},
      {'height': 2742446},
      {'height': 2742448},
      {'height': 2742450},
      {'height': 2742451},
      {'height': 2742453},
      {'height': 2742457},
      {'height': 2742460}]
```

So 2742418 is where it all began.. Lets see the transaction at that block. Copy the following into the query bar on the browser.

```
MATCH (b:Block)-[:CONTAINS]->(tx:Transaction)-[:HAS_EVENT]->(le:LogEvent)-[:CALLS]-(addy:Addres
```

If you aren't familiar with Cypher don't worry about the syntax. That query gets all the data about a single transaction on the given block. You get a visual of the transaction. You can rearrange the nodes to make them easier to see. Selecting either a node (one of the balls) or a relationship (the connections between the nodes) lets you see any properties.

1.2.1 Sovryn Protocol is created!



```
$ match pth=(b:Block)-[:CONTAINS]->(tx:Transaction)-[:HAS_EVENT]->(le:LogEvent)-[:CALLS]-(addy:Address) where b.height=2742418 return b, tx, le, addy
```

CALLS **<id>:** 2224 **name:** OwnershipTransferred **newOwner:** 0x7be508451cd748ba55dcbe75c8067f9420909b49 **previousOwner:** 0x0000000000000000000000000000000000000000
signature: OwnershipTransferred(indexed address previousOwner, indexed address newOwner)

The first transaction on the Sovryn protocol is the creation of the contract ("transfering" ownership from the null address of all zeroes).

1.3 So what? I'm here for the money

Lets chase the money. Let's analyze a high value transaction

```
[3]: q = Query()
     q.add("match (tx:Transaction) return tx order by tx.value desc limit 1")
     result = q.only()
     result
```

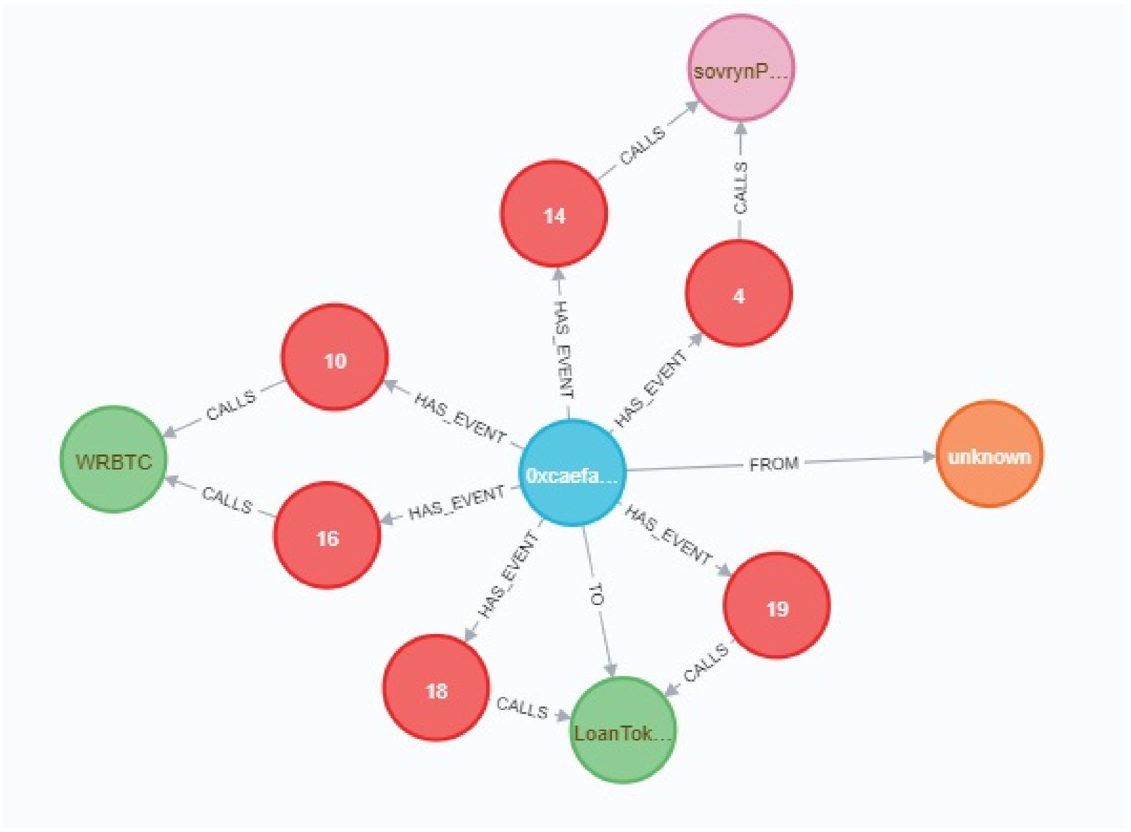
```
[3]: {'tx': {'gas_price': 60000000,
       'gas_offered': 172201,
       'gas_spent': 172201,
       'gas_quote': 0,
       'gas_quote_rate': 4083,
       'tx_offset': 4,
       'value_quote': 7350,
       'tx_hash':
     '0xcaefac99f076cd6e9e02a2b1309056eebab634f7cdf0ff28b7050dbc37c9110d',
       'value': 1800000000000000000,
       'successful': True}}
```

```
[4]: result['tx']['value']/10**18
```

```
[4]: 1.8
```

This transaction involved 1.8 wrapped BTC. Let's get more details. We use the (slightly verbose) query

```
MATCH (b:Block)-[:CONTAINS]->(tx:Transaction) where tx.tx_hash="0xbef02237efff3788082b28d74e34
WITH tx MATCH (frm:Address)<-[:FROM]-(tx)-[:TO*0..1]-(to:Address)
WITH frm,tx,to MATCH (tx)-[:HAS_EVENT]->(le:LogEvent)-[:CALLS]-(addy:Address)
RETURN tx, le, addy, frm, to
```



Looks like this transaction was a loan. Whoever owns the `from` address `0x5d0eeaeabd5123e3d557c8a552134f24c6271a74` borrowed 1.8 WRBTC. This address doesn't seem to match any Contract or Token documented as part of the sovryn protocol.

1.4 Larger scale analysis

These colorful balls are all well and good, but what if you want to analyze meaningful amounts of data. We can use the knowledge graph to do larger scale analysis as well. Let's look at a swap. We'll limit the number of results for this example, but you could just remove the `LIMIT` clause and get all the data.

[5]:
```
import plotly.express as px
from sovrynkg.swaps import get_swap_df

df = get_swap_df(skip=1000, limit=1000)
df.head()
```

[5]:
```
                   signed_at     to_amount   from_amount to_token from_token  \
0  2021-06-07 20:57:18+00:00  5.939477e+16  1.991964e+21    WRBTC       USDT
1  2021-06-07 20:49:06+00:00  3.983686e+22  1.192387e+18     USDT      WRBTC
2  2021-06-07 20:26:27+00:00  2.662208e+20  7.822917e+15     USDT      WRBTC
3  2021-06-07 20:25:50+00:00  2.485033e+20  7.302156e+15     USDT      WRBTC
4  2021-06-07 20:23:55+00:00  7.465110e+19  2.193200e+15     USDT      WRBTC
```

```
                                  smart_token          trader
0  0x30a9b9be9aa8fd64ee43b1b6c9bd474601373d34  sovrynProtocol
1  0x30a9b9be9aa8fd64ee43b1b6c9bd474601373d34  sovrynProtocol
2  0x30a9b9be9aa8fd64ee43b1b6c9bd474601373d34  sovrynProtocol
3  0x30a9b9be9aa8fd64ee43b1b6c9bd474601373d34  sovrynProtocol
4  0x30a9b9be9aa8fd64ee43b1b6c9bd474601373d34  sovrynProtocol
```

Great, we have the data. Now lets try to make sense of it. If we want to get more information about the addresses we can use a built-in tool.

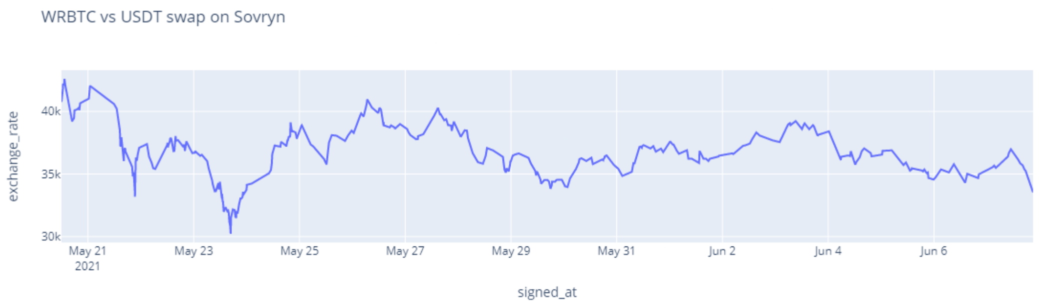
[6]:
```
import sovrynkg.contracts as contracts
wrbtc = contracts.BY_NAME['WRBTC']
wrbtc, wrbtc.address
```

[6]:
```
(<Token WRBTC:0x542…677d>, '0x542fda317318ebf1d3deaf76e0b632741a7e677d')
```

You can slice and dice your dataframe in powerful ways. Let's look at the history of the `WRBTC/USDT` swaps here.

[8]:
```
bt_pair = df[df.to_token=='WRBTC']
bt_pair = bt_pair[bt_pair.from_token=='USDT']

#both WRBTC and USDT have 18 decimals
bt_pair['exchange_rate'] = bt_pair.from_amount/bt_pair.to_amount
fig = px.line(bt_pair, x='signed_at', y='exchange_rate', title='WRBTC vs USDT␣
 ↪swap on Sovryn')
fig.show()
```



1.5 Why make this project?

In trying to learn about Sovryn I ran into two different problems.

Its hard to get data about the Sovryn protocol. Before the Knowledge Graph if you wanted data on, say, the swap prices you would have to interact with the ABIs. Now its as easy as calling a function and getting your data into python so you can start analyzing away.

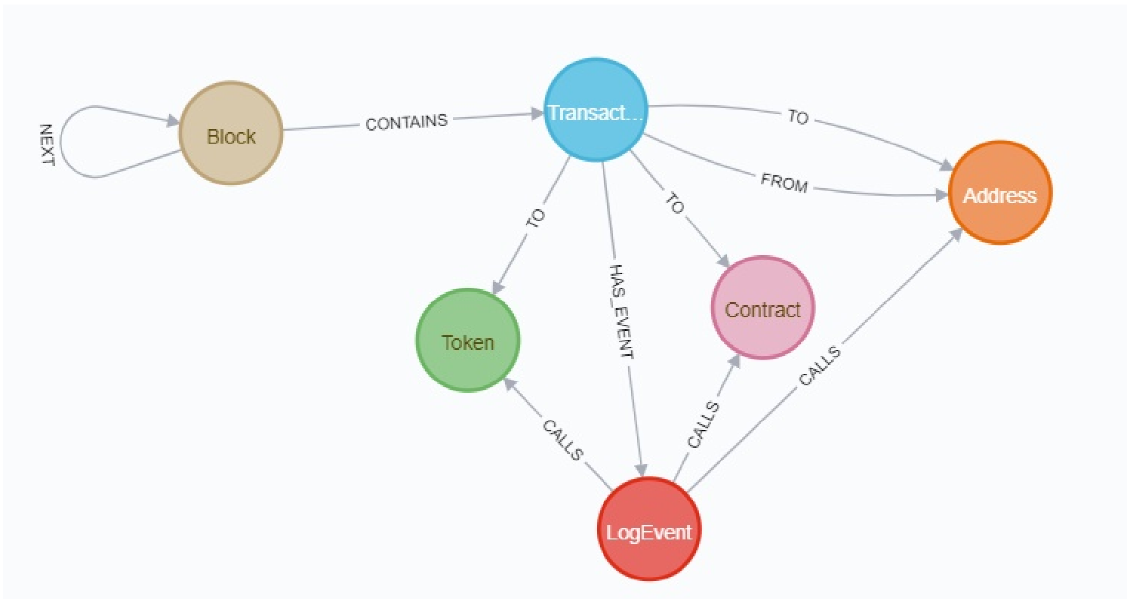
The Sovryn Protocol is complex and can be difficult to understand as a newbie. I wanted to understand more about the sovryn protocol. The Sovryn documentation was a great place to start, but it wasn't enough to get me to where I felt like I could meaningful develop on the system. There were so many moving pieces and I didn't understand how they all fit together just from the documentation. Clicking around in the RSK block explorer I found a lot of data was given as `raw_log_topics` or lengthy address strings. Being able to visualize how the various contracts and calls fit together has been very helpful.

1.6 Database Schema

The knowledge graph is made up of a few different types of nodes: `Block`, `Transaction`, `Address`, `Token`, `Contract`, and `LogEvent`. `Token` and `Contract` are subtypes of `Address`.

`Blocks` `CONTAIN` zero or more `Transactions`. The Transactions are where much of the action is. Each Transaction is `from` one `Address` and may be `to` another one. If these addresses describe known `Tokens` or `Contracts` then the information for those will be filled in.

Each Transaction has one or more `LogEvents`. Each of these events `CALLS` various Addresses (or Tokens, or Contracts). In creating this knowledge graph, a number of ABIs were parsed so that the information in each of the `CALLS`



1.7 Future Development

There's a lot that can be built from here. Some immediate to-dos are

- Packaging of more queries. Not everyone wants to learn the Cypher query langauge, so having common operations packaged up as functions would be very helpful. See the `swaps` module for an example.

- Hosting with automatic updates. Moving from docker hub to a hosted api service would make it even easier to use and allow for up-to-the-blockheight updates.

2 Stay Sovryn!