Adamnite Serialization and Merkle Trees

Adamnite Serialization is a process for serializing data structures for use within the Adamnite Protocol. It defines mappings for these data structures to byte sequences for easier transmission and fetching of data. Adamnite Serialization will be the primary serialization method used in the Adamnite Blockchain, with definitions for accounts, messages, transactions, and blocks all using Adamnite Serialization to store and transmit data. This document contains a formalization of Adamnite Serialization, with definitions for key mappings to byte arrays and an example implementation in Python. Adamnite Serialization is a work in progress, and may evolve according to need as the Adamnite Protocol itself changes.

Definition

The Adamnite Serialization Protocol will mainly be applied to lists and strings. Strings are defined as byte arrays, while lists are simply collections of strings and other items. For a single byte, like RLP, the byte is its own encoding. For strings with byte length between 0 and 64, the encoding value is 0x70 plus the length of the string in bytes, followed by the string split into individual characters itself. For strings longer than 64 bytes, the encoding will be 0xa1 plus the length of the string in binary format, followed by the length of the string. For lists with a total payload of less than 64 bytes, the encoding will be 0xco plus the length of the list followed by the encodings of the individual items in the list. For lists with a total payload of more than 64 bytes, the encoding will be 0xfo plus the length in bytes of the payload, followed by the length of the payload, followed by the concatenation of the encodings of the list.

```
def encoding(input):
      if isinstance(input,str):
      length = len(input)
      if length == 1:
            return input
      else:
            return encode_length(length,0x70) + in
      elif isinstance(input,list):
      output = ''
      for item in input:
            output += encode(item)
      return encode_length(len(output),0xc0) + output
def encode_length(item,offset):
      if L < 64:
      return chr(item + offset)
```

```
      elif L < 256**8:
      BL = to_binary(L)
      return chr(len(BL) + offset + 63) + BL
      else:
      raise Exception("input too long")

def to_binary(x):
      if x == 0:
      return ''
      else:
      return to_binary(int(x / 256)) + chr(x % 256)
```

Data Storage (Implementation of Merkle Trees)