Storing and Retrieval of Structured Data on Blockchain with BlockChi and Ethereum

Author: Boyan Yankov

Abstract: BlockChi creates an easy way of storing and retrieving data on the blockchain. It is a technology that solves some of the current limitations of blockchain storage and public blockchain networks that have transaction payload support. It allows parties to author, publish, extend and access data records in a decentralized, immutable and secure manner, supports complex data types, such as structured data and media, supports metadata to allow anonymously, identified or trusted publishing. BlockChi format is compatible with various blockchain networks. Data is presented as a JSON object, encoded in hexadecimal format and contained in the payload of blockchain transaction(s). There is an Ethereum implementation – BlockChiEth, which is a JavaScript class that allows frontend and backend applications (DAPPs) to interact with Ethereum blockchain.

Keywords: blockchain, data storage, Ethereum, DAPP, decentralized

JEL C880

INTRODUCTION

Blockchain technology is a significant advancement in achieving consensus between two or more parties without relying on a trusted third party [1]. Blockchain has become widely used as a ledger for storing transactional data, i.e. in Bitcoin and other cryptocurrencies and is becoming popular in private blockchains and business use cases [2][3]. Other uses also exist, such as smart contracts, tokens and decentralized storage of data [4][5][6][7].

BlockChi introduces a new approach to storing and retrieving data on a blockchain that allows parties to author, publish, extend and access data records in a decentralized, immutable and secure manner [8][9]. BlockChi Ethereum is an Ethereum implementation of the BlockChi technology for storing and retrieving data on the blockchain. BlockChi also supports complex data types, such as structured data and media; metadata to allow anonymously, identified or trusted publishing; publishing large files, etc.

Technically, the data is presented as a JSON document (object) [10], encoded in hexadecimal format and contained in the payload of blockchain transaction(s), thus ensuring human readability and interoperability.

STORING AND RETRIEVING DATA ON BLOCKCHAIN WITH BLOCKCHI

BlockChi is a technology for storing and retrieving data on a blockchain. The data is presented as a JSON object (as described by a JSON Schema), contained in the payload

of a blockchain transaction.

Writing data to the blockchain is done by preparing the data as a JSON object, according to the provided JSON schema, converting the JSON object to hexadecimal format and sending a transaction(s) from the wallet of the author with the data as transaction payload.

Reading data from the blockchain is done by reading the transaction(s) and getting the payload, decoding the payload from hexadecimal to text and parsing as a JSON object, validating against the JSON schema and displaying the data contained in the object.

In some blockchain networks, there are limitations он the maximum size of a single transaction. For example, in Ethereum the transaction size is limited indirectly by the block gas limit. The transaction size limit directly limits the maximum amount of data stored in a single transaction, which makes storing a large file in a transaction impossible. To overcome data size limits, BlockChi first converts data to a JSON object and then splits the object into multiple hexadecimal chunks of data, that can be stored in multiple transactions within their data limit. BlockChi can then read all transactions, combine the data and decode it from hexadecimal to JSON which results in the original JSON object.

In Ethereum [11] the transaction payload is contained in the “input” field of the transaction which BlockChi uses to store data (Table 1). Some of the other transaction fields can also have a special meaning for BlockChi.

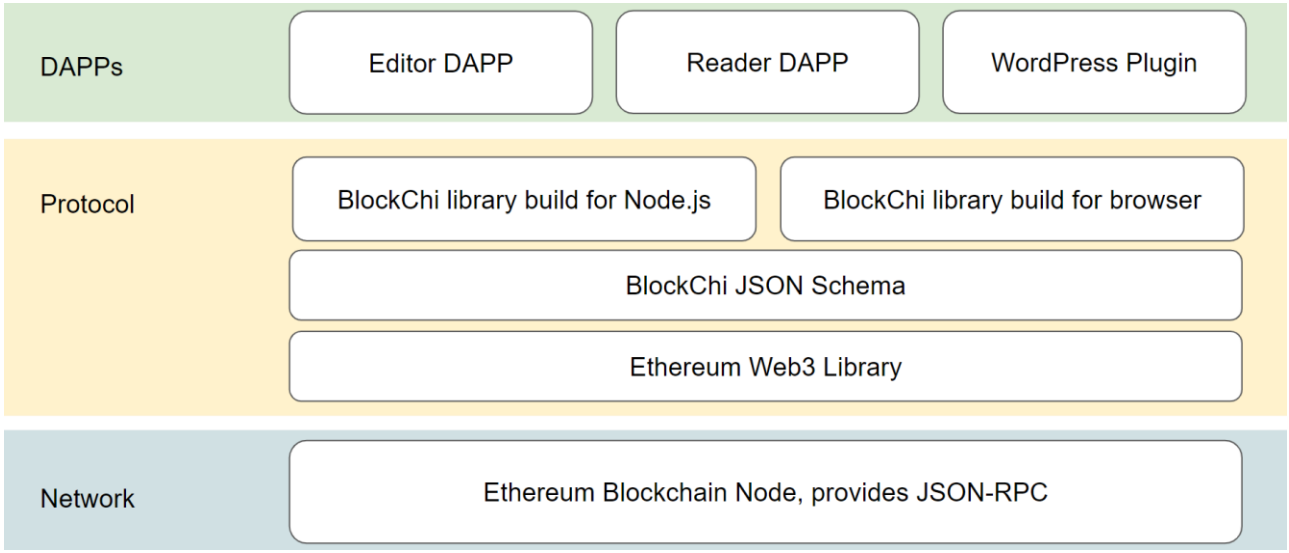
Table 1. Integration of BlockChi with an Ethereum transaction

Ethereum Transaction Field	Default Field Meaning	Field Meaning for BlockChi
hash	Hash of the transaction	Default
nonce	The number of transactions made by the sender before this one	Default
blockHash	Hash of the block where this transaction was in. null when it’s pending.	Default
blockNumber	Block number where this transaction was in. null when it’s pending.	Default
transactionIndex	An integer of the transactions index position in the block. null when it’s pending.	Default
from	Address of the sender.	Unique identifier of the address of the data author
to	Address of the receiver. null when it’s a contract creation transaction.	Unique identifier of the address of the data publisher
value	Value transferred in wei.	Data publishing priority / Default
gasPrice	Gas price provided by the sender in wei.	Default
gas	Gas provided by the sender.	Default

input	The data sent along with the transaction.	Published content presented as a JSON object encoded in hexadecimal format.

BlockChi works on top of the network layer (Fig. 1), which can be the Ethereum blockchain network or other blockchain networks that support transaction payload. The bottom layer is the network node which provides JSON -RPC interface. BlockChi protocol layer connects to the blockchain node and consists of web3 library, BlockChi JSON schema, and BlockChiEth library – build for JavaScript - Node.js and browser versions. Decentralized applications (DAPPs) are the topmost layer. They use an instance of BlockChiEth library to interact with the blockchain network.

Fig. 1 Layered model of BlockChi:



USE CASES OF BLOCKCHI

BlockChi has been successfully applied in content publishing and data storage [14]. It has use cases in scenarios that take advantage of blockchain technology benefits for storing data and publishing information. Key benefits are: decentralized, immutable, secure, the extendibility of data, digital economy features, support of complex data types, support of structured data and media, support of anonymous, identified or trusted publishing, support of publishing large files. Use cases include:

- Storing or publishing documents: company archives, laws, declarations, contracts, etc. An author can be specified declaratively and/or the author can be verified. Documents can be self-published or published by a third-party publisher for free or in exchange for a reward. The published documents can be extended with amendments and corrections.
- Publishing articles and books. Documents can be authored anonymously or an author can be specified, and/or the author can be verified. Documents can be self-published or published by a third-party publisher for free or in exchange for a reward. The published documents can be extended with chapters and

amendments.

- Publishing announcements: personal feed (log), newsletter, social sharing, marketing feed, etc.
- Storing or publishing media content: images, audio, video, etc. Content can be published for free publishing or with a reward scheme.
- Storing data, accessible for smart contracts is a possible future use.
- Publishing applications is a possible future use.

WORKFLOW AND SOFTWARE APPROBATION

The proposed workflow for writing and retrieving data with BlockChi has been validated to work with the Ethereum network. The workflow consists of the following steps:

- Prepare the transaction payload in JSON format. It can be easily done by using any JSON editor or the provided BlockChi editor [12].
- Convert transaction payload to hexadecimal format as required by Ethereum blockchain. Done automatically by the provided BlockChi editor, or can be done with BlockChiEth JavaScript library.
- Send an Ethereum transaction by using a wallet or from your application. MyCrypto wallet has been used during tests [13]. From your address (used as author) to another address (used as published). In the data/payload field use the prepared transaction payload - in hexadecimal format. For testing purposes, a testing network can be used, for example, the Ropsten testnet network.
- Read transaction data. Use the provided BlockChiEth JavaScript library to read and visualize transaction data.

BlockChi JSON objects need to be structured according to the BlockChi JSON schema to be considered valid. Reader and writer applications implement support of the schema which ensures their interoperability. The below code shows the current BlockChi JSON schema as published on GitHub.

BlockChi JSON schema

```
{
  "definitions": {},
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://example.com/root.json",
  "type": "object",
  "title": "BlockChi JSON Schema",
  "default": null,
  "required": [
    "title",
    "items"
  ],
  "properties": {
    "title": {
      "type": "string",
      "title": "Document title",
      "default": "",
      "examples": [
        "The holy bible"
      ],
      "pattern": "^(.*)$"
    },
```

```
    "description": {
      "type": "string",
      "title": "Document description",
      "default": "",
      "examples": [
        "Containing the Old and New Testaments"
      ],
      "pattern": "^(.*)$"
    },
    "parent": {
      "type": "string",
      "title": "Document parent, transaction hash",
      "default": "",
      "examples": [
        ""
      ],
      "pattern": "^(.*)$"
    },
    "author": {
      "type": "string",
      "title": "Document author",
      "default": "",
      "examples": [
        "John Smith"
      ],
      "pattern": "^(.*)$"
    },
    "authorUri": {
      "type": "string",
      "title": "Document author URI",
      "default": "",
      "examples": [
        ""
      ],
      "pattern": "^(.*)$",
      "format": "uri"
    },
    "validFrom": {
      "type": "string",
      "title": "Document valid from, date",
      "default": "",
      "examples": [
        ""
      ],
      "pattern": "^(.*)$",
      "format": "date-time"
    },
    "validUntil": {
      "type": "string",
      "title": "Document valid until, date",
      "default": "",
      "examples": [
        ""
      ],
      "pattern": "^(.*)$",
      "format": "date-time"
    },
    "items": {
      "type": "array",
      "items": {
        "type": "object",
        "title": "item",
        "properties": {
          "title": {
```

```
            "type": "string",
            "title": "Item title",
            "default": "",
            "examples": [
              "Genesis"
            ],
            "pattern": "^(.*)$"
          },
          "data": {
            "type": "string",
            "title": "Item data, text",
            "default": "",
            "examples": [
              "God creates this earth and its heaven and all forms of life in
six days.."
            ],
            "pattern": "^(.*)$"
          },
          "dataUri": {
            "type": "string",
            "title": "Item data, file in data URI format",
            "default": "",
            "pattern": "^(.*)$",
            "media": {
                "binaryEncoding": "base64"
            }
          },
          "dataUriExternal": {
            "type": "string",
            "title": "Item data, external URL",
            "default": "",
            "examples": [
              ""
            ],
            "pattern": "^(.*)$"
          },
          "dataHash": {
            "type": "string",
            "title": "Hash of item data, md5",
            "default": "",
            "examples": [
              ""
            ],
            "pattern": "^(.*)$"
          }
        }
      }
    }
  }
}
```

For a JSON object to be valid a BlockChi object, it needs to be both syntactically valid and to validate against the BlockChi JSON schema. The below code is an example of a basic valid object.

A basic example of BlockChi JSON object

```
{
  "title": "The Gift of the Magi",
  "author": "O. Henry",
  "description": "Genre: fiction",
  "items": [
```

```
    {
      "title": "1",
      "type": "text/plain",
      "data": "One dollar and eighty-seven cents. That was all. And sixty cents
of it was in pennies. Pennies saved one and two at a time by bulldozing the
grocer and the vegetable man and the butcher until one's cheeks burned with the
silent imputation of parsimony that such close dealing implied. Three times
Della counted it. One dollar and eighty-seven cents. And the next day would be
Christmas.       There was clearly nothing left to do but flop down on the
shabby little couch and howl…"
    },
    {
      "title": "2",
      "type": "text/plain",
      "data": "There was a pier-glass between the windows of the room. Perhaps
you have seen a pier-glass in an $8 Bat. A very thin and very agile person may,
by observing his reflection in a rapid sequence of longitudinal strips, obtain a
fairly accurate conception of his looks. Della, being slender, had mastered the
art…"
    }
  ]
}
```

The BlockChiEth class instance allows developers to establish a connection to an Ethereum node and to read the transaction data. Then the transaction data is converted to JSON object which is validated according to the JSON schema. The data is extracted from the JSON object (texts, attached files, etc.) and can be used in the application (DAPP). The below code demonstrates a basic example of reading data from the Ethereum blockchain with JavaScript, Node.js and BlockChiEth library.

Reading data from the Ethereum blockchain

```
'use strict';
//this is an example for node.js
console.log('BlockChi - Blockchain Payload Tools Demo');

const BlockChiEth = require('./src/blockchieth');

for (let j = 0; j < process.argv.length; j++) {
    console.log(j + ' -> ' + (process.argv[j]));
}

//first argument - Ethereum HTTP API url here or test the example
//default value
  let apiUrl = "https://ropsten.infura.io/"
if (process.argv[2] !== undefined){
  let apiUrl = process.argv[2];
}

//second argument - Ethereum transaction hash here or test the exam
//default value
  let transactionHash =
'0x243b483b8f3aa0d22a732f1075eccfe499d4dd535d45a2c0858d560ffd2f6c83';
if (process.argv[3] !== undefined){
  let transactionHash = process.argv[3];
}

const blockChiEth = new BlockChiEth(apiUrl);

console.log('Getting transaction data...');
```

```
let transactionPayload = blockChiEth.getTransactionPayload(transactionHash);

console.log('Transaction data, raw (JSON / text):');
console.log(transactionPayload.payloadAscii);

let transactionPayloadObject =
blockChiEth.textToJson(transactionPayload.payloadAscii);

console.log('Transaction data, decoded:');

console.log('Title: '+transactionPayloadObject.result.title);
console.log('Description: '+transactionPayloadObject.result.description);
console.log('Author: '+transactionPayloadObject.result.author);
console.log('Network: '+apiUrl);
console.log('Transaction: '+transactionHash);

for (let itemNumber = 0; itemNumber <
transactionPayloadObject.result.items.length; itemNumber++) {
  let item = transactionPayloadObject.result.items[itemNumber];

  console.log('Item title: '+item.title);

  if (item.data !== undefined && item.data != ''){
    console.log('Item data: '+item.data);
  }

  if (item.dataUri !== undefined && item.dataUri != ''){
    let dataType = item.dataUri.split(';')[0];
    let mimeType = dataType.split(':')[1];

    if (mimeType == 'image/png' || mimeType == 'image/jpeg'){
      console.log('(image attachment)');;
    } else {
      console.log('Error: data type '+mimeType+' is not supported by the
reader!');
    }
  }
}
```

To write data to Ethereum blockchain, first it the data needs to be prepared as a JSON object. This can be done in any JSON editor by loading the provided JSON schema. One example is to use the default web-based BlockChi editor. The JSON object will contain all data, including any files. The next step is to prepare the data for writing – in hexadecimal format. The web-based BlockChi editor does this automatically and also supports splitting large data into chunks. The BlockChiEth library can also be used for preparing the data. And the final step is to send the transaction(s) with data as payload. The below example demonstrates how the JSON object can be validated against the BlockChi JSON schema and prepared for writing on Ethereum blockchain with JavaScript, Node.js and BlockChiEth library.

Validating JSON object and preparing transaction payload

```
//this is an example for node.js
const BlockChiEth = require('./src/blockchieth');
const fs = require('fs');

console.log('BlockChi - Blockchain Payload Tools Demo');

console.log('Validate JSON payload and convert to hex');
```

```
const blockChiEth = new BlockChiEth();

console.log('Loading schema...');
let schemaRawData = fs.readFileSync('./schemas/default-schema.json', "utf8");
let schema = blockChiEth.textToJson(schemaRawData);

console.log('Loading data...');
let operationInput = fs.readFileSync('./schemas/sample-data.json', "utf8");

//validate data
console.log('Validating JSON:');
let errorString = '';

let operationInputJson = blockChiEth.textToJson(operationInput);

if (operationInputJson.result){
  let validationResult = blockChiEth.validateJson(schema,
operationInputJson.result);

  errorString += 'Validation result: '+validationResult.result+' ';
  if (validationResult.errors){
    errorString += ' Validation errors:
'+blockChiEth.jsonToText(validationResult.errors);
  }
} else {
  errorString += 'JSON errors: '+operationInputJson.errors;
}
console.log(errorString);

//convert data to hex
console.log('Converting to HEX:');
let operationOutput = blockChiEth.toHex(operationInput);
console.log(operationOutput);

console.log('To store the document on blockchain, use the HEX value as
transaction payload!');
```

CONCLUSION

BlockChi is innovation - it solves some of the current limitations of blockchain storage and public blockchain networks that have transaction payload support. These are: no popular software for using the transaction payload for storage, lack of standards for storing and retrieving data, lack of basic features (such as storing various data formats, metadata), transaction size limits. BlockChi can be used in both decentralized applications' backend (with NodeJS) and frontend (in a web browser). BlockChi has been successfully applied in content publishing and data storage on a testing web site.

The current state of blockchain technology has many limitations and some of them affect BlockChi. The main challenges to resolve are:

- Blockchain is an emerging technology and the biggest challenge is its adoption in real-life use cases.
- Storage capacity and price – blockchain offers high-cost storage. Data is typically replicated on all nodes, which is expensive. Public blockchain solutions offer a high cost of storage and free access. Private blockchain networks are highly configurable in terms of the number of nodes and storage costs can be optimized. There are also blockchain solutions optimized for storage efficiency.

- Some operations are not as efficient as in relational databases. For example, getting the list of transactions for an address on a blockchain is a trivial task which typically consists of iterating back from the current block until all transactions for the address have been discovered (or until the first block). However, the performance of executing is often not satisfactory for real-time use cases. There are many possible solutions to this problem, including on-chain solutions (smart contract, special blockchain implementation) and off-chain solutions (creating an index, caching).
- Public and private blockchain networks are often unregulated by government authorities and offer a high level of anonymity. This creates the possibility for illegal use – for storing, publishing and distributing of restricted types of data.

BIBLIOGRAPHY

1. Swan, M., 2015, Blockchain: Blueprint for a new economy, books.google.com,

2. Nakamoto, S., 2008, Bitcoin: A peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf

3. Ethereum Project, 2019, https://www.ethereum.org/

4. SAP HANA Blockchain: An introduction | SAP Blogs, 2018, https://blogs.sap.com/2018/06/13/sap-hana-blockchain-a-technical-introduction/

5. Omohundro, S., 2014, Cryptocurrencies, smart contracts, and artificial intelligence, AI matters, dl.acm.org

6. Wilkinson, S., Lowry, J., Boshevski, T., 2014, Metadisk a blockchain-based decentralized file storage application“, storj.io

7. Introduction - Swarm 0.5 documentation, 2019, https://swarm-guide.readthedocs.io/en/latest/index.html

8. BlockChi Ethereum - Blockchain Payload Tools Website, 2019, https://boyanborislavovyankov.github.io/BlockChi-Ethereum-Blockchain-Payload-Tools/

9. BoyanBorislavovYankov / BlockChi-Ethereum-Blockchain-Payload-Tools code repository on GitHub, 2019, https://github.com/BoyanBorislavovYankov/BlockChi-Ethereum-Blockchain-Payload-Tools

10. JSON Schema, 2019, https://json-schema.org/

11. web3.js 1.0.0 documentation, 2019, https://web3js.readthedocs.io/en/v1.2.0/web3-eth.html#gettransaction

12. BlockChi Editor | Satoshi Journal, 2019, https://satoshijournal.com/blockchi-editor/

13. MyCrypto is an open-source, client-side tool for generating ether wallets, 2019, https://mycrypto.com/

14. Satoshi Journal | Blockchain publishing made easy, 2019, https://satoshijournal.com/