Kevacoin: A Peer-to-Peer Key-Value Database

The Kevacoin Project (DRAFT)
info@kevacoin.org
www.kevacoin.org

Abstract. Kevacoin is a peer-to-peer key-value database built on Bitcoin's proven consensus mechanism and codebase. It is a public, permissionless and general purpose database that everyone can read and write on. We introduce the concept of namespace to avoid the problem of name squatting and collision. The database can be accessed through JSON-RPC APIs without smart contract programming. As a decentralized database, it is suitable for various kinds of decentralized applications.

1. Introduction

With the explosive growth of internet users and applications, a trend is clearly emerging. Data has become a very valuable resource and can be used to drive a profit for the owners through various means, for example, optimization of business and targeted advertising. Big corporations aggressively collect and monopolize user data. With monopolization comes the censorship of data, as they control what can and cannot be published. In many cases, users do not know what data has been collected and whether the entity who collects the data will use them responsibly.

What is needed is a database that is not owned by any person or organization, allowing anyone to publish data that is freely available to everyone, without the concern of being withheld or being monopolized for the purpose of monetization. This database must not be entrusted to any individuals or organizations but should be based on cryptographic proof of individual data ownership. Such a system exists for the ownership of electronic coins (cryptocurrency), the first and the most important one being Bitcoin [1]. Bitcoin operates a peer-to-peer ledger (blockchain) that tracks the creation and transactions of electronic coins. The ledger is the database of the coins. Here we show that it is possible to expand the use of this database to support arbitrary data

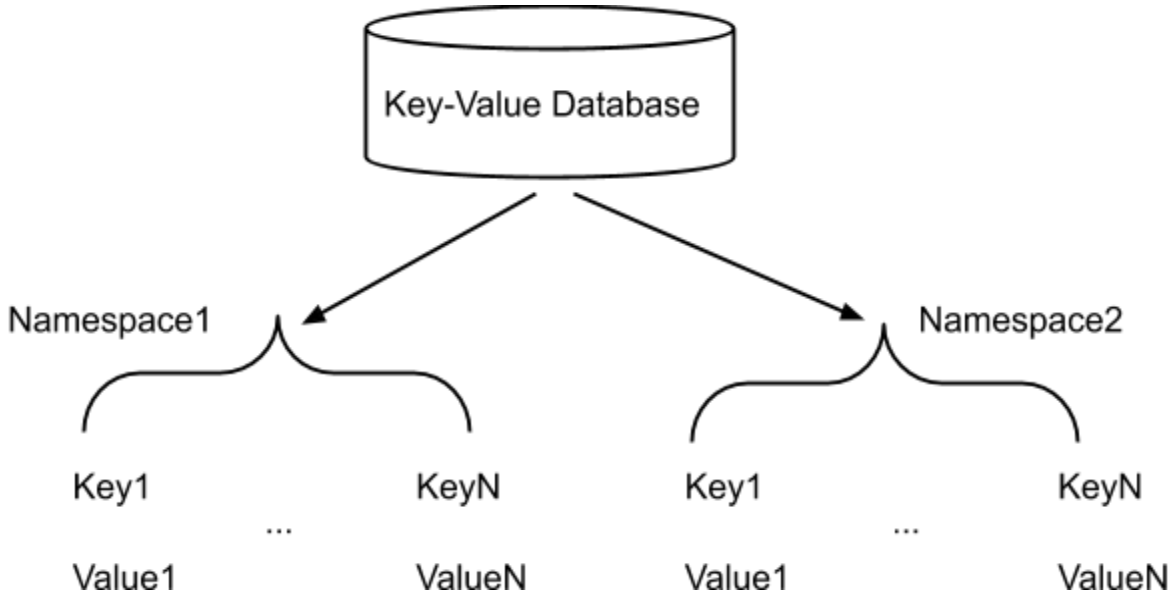
with cryptographic proof of ownership and allow the transfer of ownership. At the same time, the creation, update, retrieval, and transfer of the data is as easy as transacting the coins.

2. Key-Value Database

Although Bitcoin ledger was not designed as a general purpose database , its OP_RETURN opcode allows one to add a small amount of arbitrary data (83 bytes) to the ledger. Some applications exploit this feature to use Bitcoin's ledger as a database in an unwieldy way. However, it is well recognized that a structural data model will make it much easier to create, update, search, and retrieve data. A database is such a tool to store and organize data in one of the well defined models: relational, document, object, and others. One of the simplest yet commonly used databases is the key-value database, which provides a dictionary data storage. Data is added to the dictionary with unique keys, and can be retrieved or updated through the keys. The simplicity of the key-value database makes it an ideal candidate to implement on the blockchain.

A public and permissionless blockchain allows anyone to write to it. It is undesirable to allow anyone to arbitrarily update the key-value pairs as this will create a chaotic situation and make it impossible to develop applications with the database. We may assign the ownership of a key to the one who first creates it, and only they can update the value of the key. The owner has the option to explicitly transfer the key to other people. However, this makes the key name a potentially valuable asset, especially shorter names or names associated with brands. This is a phenomenon called "name squatting" and is familiar to Namecoin [2] [3], the domain name registration coin. The uniqueness of the key name also makes it hard to develop on the database as the developer cannot freely assign a name to a key as the name may have been taken by someone else.

We introduce the concept of the namespace to solve these issues. A namespace is represented by a network-unique, base58-encoded SHA256 hash value of a previous transaction ID. The creator of the namespace is the owner and is free to insert or update the key-value pair in the namespace.



Bitcoin node uses LevelDB as a database to store block and transaction information. LevelDB is a key-value database, so it is convenient to also use it to store our newly introduced key-value pairs.

Cryptocurrencies that support smart contracts provide data storage and access on the blockchain (e.g. Ethereum). Writing and reading the data requires smart contracts, which are written in the scripting languages specific to the platform (e.g. Solidity for Ethereum). Implementing secure smart contracts is a difficult task [4] [5], and it is almost impossible to update the contracts once they are published. The goal of Kevacoin is to provide a general purpose, decentralized database that can be accessed through simple APIs in any programming language. To that end, a small set of JSON-RPC APIs is added to extend Bitcoins' existing APIs.

3. Opcodes for the Database

In Bitcoin, the ownership transfer of coins is represented in a transaction with inputs and outputs. The inputs are the previous transactions, and the outputs are the instructions on how to send the coins, including the amount and code written in Bitcoin script on where to send the amount (scriptPubKey). The owner must digitally sign the transaction using his private key to prove that

he is the receiver of the previous transaction output, and confirm the new owner who will receive the coins.

The Bitcoin script has an opcode OP_RETURN that allows one to append 83 bytes of arbitrary data. This opcode is not a good candidate to handle the data due to its *ad hoc* nature and small data size. We introduce several new opcodes to support the operation of the database.

Opcode	Description
OP_KEVA_NAMESPACE <internal_name>	Create a namespace with the given internal name. The internal name is for the user's internal use and does not need to be unique.
OP_KEVA_PUT <namespaceId> <key> <value>	Insert or update a key value pair in the given namespace. The maximum key size is 255 byte, and the maximum value size is 3072 bytes.
OP_KEVA_DELETE <namespaceId> <key>	Delete the key-value pair in the given namespace.

The first time a namespace is created, a small amount of coin (0.01) is reserved for the namespace. This namespace coin cannot be spent for normal transactions, and will not show up in the wallet balance.

To create a namespace, one sends 0.01 coin to himself, with the scriptPubKey in the transaction output prefixed with OP_KEVA_NAMESPACE:

OP_KEVA_NAMESPACE <namespaceId> <internal_name>

A node understands the transaction will perform the following actions:

1. Remove the prefix and handle the rest of the pubScriptKey as normal.
2. Create an entry in LevelDB.

To insert a key-value pair in the namespace, one sends the above namespace coin to himself, with the scriptPubKey prefixed with OP_KEVA_PUT:

OP_KEVA_PUT <namespaceId> <key> <value>

A node understands the transaction will perform the following actions:

1. Remove the prefix and handle the rest of the pubScriptKey as normal.
2. If the key does not exist, insert the key-value pair.
3. If the key already exists, update the value associated with the key.

One can repeatedly use OP_KEVA_PUT to create new key-value pairs or update the existing ones. The maximum key size is 255 bytes, and the maximum value size is 3072 bytes.

The OP_KEVA_DELETE is similar to OP_KEVA_PUT, except that it assigns an empty value to the key.

The network checks the namespace ID in the transaction input and the one in the output. It will reject the transaction if they do not match.

A user does not need to write a script using the new opcodes to access the database. We provide a small set of JSON-RPC APIs that can be called by command lines, GUI and any programming languages.

4. Namespace Ownership

One can transfer the ownership of the namespace to the next one by creating an OP_KEVA_PUT transaction, with the next owner's receiving address in the scriptPubKey. After the transaction, the next owner will be able to insert or update key-value pairs in the namespace.

A MultiSig address allows a namespace to be controlled by multiple parties, that is, the transaction must be signed by M-of-N keys in order to update key-value pairs in the namespace. This is a security feature that can be used when multiple parties' approvals are required to update the data.

5. Notification of Data

Key-Value data structure is not always the optimal way to organize and query data, especially when the key-value pairs are in different namespaces. We provide a ZeroMQ notification API to listen to the data that is being written to the database. This ZeroMQ API is in addition to the existing Bitcoin ZeroMQ APIs for block and transaction events. Developers can use this API to monitor and extract the data that are of interest to them, and export them to a separate database (e.g. an SQL database) to make it easy to query and retrieve data later.

For example, in a decentralized Twitter-like application, users would write their messages in their respective namespaces. The convention is that if the key starts with "microblog_", its value contains a message. Through the notification API, once the key with the given prefix comes in, its value (i.e. the message) is written to another database that supports a more sophisticated data model (e.g. an SQL database). This makes it easy for the application to perform searches of the messages or retrieve the messages of a user.

6. Data Security

Since the key-value database is based on Bitcoin's proven consensus protocol, the database is guaranteed to be consistent and available across all nodes. This characteristic differentiates it from the peer-to-peer distributed file system like Interplanetary File System (IPFS) [12]. Data in

IPFS may not be available if the nodes that serve the data become unavailable. A database implemented on top of IPFS will have the same availability issue [6].

All the key-value pairs in a transaction included in a block are cryptographically verified and are tamper-proof. One of the major concerns and risks of Proof-of-Work based cryptocurrencies is 51% attacks. However, the attacker cannot change the key-value pairs in other people's namespaces as they are protected by the signatures signed by the owners' private key. Key-value pairs tampered by non-owners can be easily detected by any nodes and will not be accepted by the network. The worst an attacker can do is to block or reverse the key-value transactions. For example, miners with 51% mining power can conspire to exclude certain key-value pairs from being included in the blocks.

If the 51% attack is temporary, the key-value pairs in the memory pool will eventually be included in a block as soon as the attack stops. If the miners conspire to exclude certain data from certain sources for the long term, the cryptocurrency will be known for its state of being constantly attacked and lose its attraction and its value. The miners will suffer as a result.

7. Applications

7.1. Decentralized Microblogs

In traditional microblogging applications, e.g. Twitter, the clients send messages to the server, which stores the messages in a central database, and dispatches the messages to the followers. The application requires both a central database and a central server. The server has the full right to decide whether to store and dispatch a message.

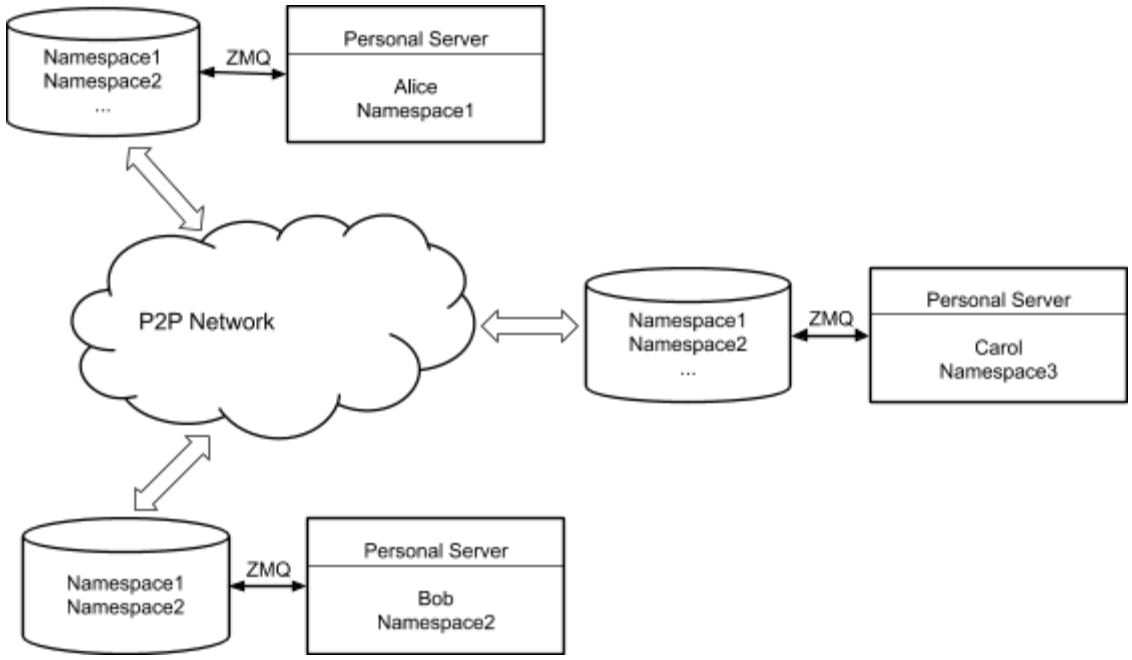
In a decentralized microblogging application, users have total control of their data. They write the message in their own namespace and also maintain a list of namespace IDs of the people who they follow. The fundamental difference from the traditional centralized model is that no central authority is coordinating the writing of data to a central database. Users participate in

microblogging by writing to a namespace with a specific convention. An application that recognizes the format will process and distribute the content. Below is a very basic example of a convention to demonstrate how a decentralized microblog would function.

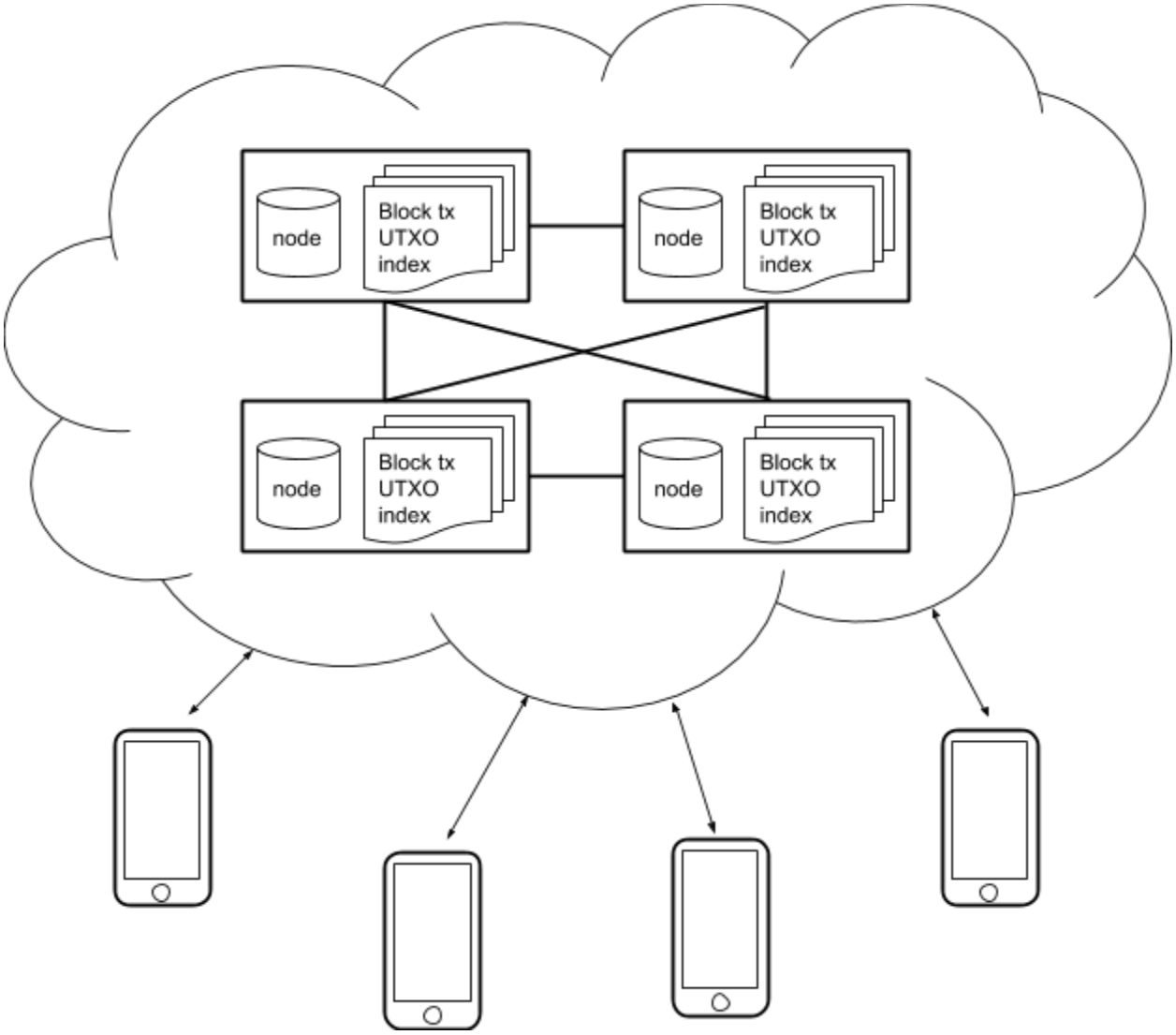
Alice wants to participate in decentralized blogging (dblog). Instead of signing up for an account on a website run by a corporation, she creates a namespace "Namespace1" on the decentralized database. The community has developed a data format "dblog_v1.0" for anyone who wants to participate in blogging. Alice decides to join the community, so she indicates this by adding a key "Blog_format" with the value "dblog_v1.0" and organizing her data according to the format. Alice adds her nickname by adding the "Nickname" key, and follows a friend Bob by adding his namespace ID in value of the "Following" key. She starts to publish her blogs by adding keys "Msg1", "Msg2" with the corresponding content.

The dblog application can be in the form of a personal server running on the user's computer or a third-party server which provides service to users. A user chooses a personal server to have full control of what the user wants to read or write. A user can choose a third-party server for convenience with the possible sacrifice of full control, e.g. the third-party server may filter certain messages before delivering to the users. This is a problem in the traditional centralized blog service because it has complete control of the data and no other entities can provide the same service. With dblog, anyone can provide a service with the publicly available data and users are free to choose the service that satisfies their needs. In this case, the dblog service can even offer message filtering as a desirable service as some users do not want to see certain messages.

With the personal server, a user runs both a node and a personal server. In the case of Alice, her personal server knows the people she follows, and subscribes to data events of their namespace using the ZMQ API. The server notifies Alice when the events arrive, e.g. someone adding a new message.



The personal server requires a full node which is not suitable for mobile phones, as they do not have the constant network connection to synchronize with other nodes, as well as having limited storage. Instead of relying on a third-party service, they can connect to the trusted public nodes to get information about the blockchain. The public node can either be a full node, or a service node (e.g. Electrum server [11]) that provides APIs to query blockchain information by indexing the blocks, transactions, and the balance of addresses in a database. An electrum server can be expanded to provide APIs to query key-value databases and other database related services. These public nodes are unlike the traditional servers in that they are configured to support peer discovery and form a peer-to-peer network. Similar to the P2P cryptocurrency network, an application connects to peers in the network through peer discovery. Unlike the traditional central servers, anyone can set up and run servers and join the peer-to-peer network to share the load of the network and support more users.



Finally, with the third-party server, users do not need to be concerned about nodes and servers. However, they still write to their own namespaces and maintain the list of people they follow, that is, they still have full control of their own data. The third-party server works similarly to the personal server, except that it is responsible for dispatching messages to respective users. The third-party server can provide additional services, such as content recommendation, search, push notifications and filtering as desired by the users. Users have the freedom to choose any third-party servers they like, without account and data migration.

One desirable feature of microblog is the instant notification of messages. A message is theoretically in the key-value database when it is validated and included in a block. However, it

is not necessary to wait for the confirmation before showing the messages to the user. The messages in the memory pool propagated by the nodes are cryptographically validated by their digital signature and can be trusted that they are from the ones who own the private key. By using the messages in the memory pool, the application enjoys almost instantaneous notifications. This also serves as an effective defense against the censorship of the miners if they conspire to discard or rollback certain messages. People will receive messages as soon as they are propagated and show up in the memory pool, even before they are included in the blocks. When certain kinds of messages are received but not included in the block later, people will detect the possibility of censorship by the miners.

7.2. Decentralized Digital Identity

The Kevacoin's decentralized key-value database is particularly suitable as a public infrastructure to enable verifiable, decentralized digital identity. W3C, the international standards organization for the World Wide Web, has a working draft on DID [7]. The DID infrastructure calls for a global key-value database, which can be DID compatible public blockchains, distributed ledgers, or file systems. A list of proposals on DID support on several blockchains and distributed file systems can be found in [8]. For example, Bitcoin has a specific proposal [9]. The Bitcoin proposal is primarily based on the *ad hoc* nature (and 83 bytes storage) of OP_RETURN opcode and tries to simulate the work of a key-value database in a way that is not originally intended for Bitcoin. The same proposal can be greatly simplified and naturally implemented in Kevacoin. Developers can provide DID as an integrated feature of Kevacoin wallet by calling a set of simple APIs to access the underlying key-value database. It can be argued that Kevacoin is exactly the infrastructure needed for public identifiers.

W3C lists a variety of use cases [10] for the decentralized identifier, ranging from enterprise, education, healthcare, law, and security (Single Sign-On). The Single Sign-On (SSO) is a feature that can be particularly appealing to many internet users today. SSO with a public identifier eliminates the use of passwords (which can be easily misused, forgotten, or stolen) for different websites, and replaces it with public key authentication. A mobile Kevacoin wallet with DID support is ideal for this purpose.

SSO can be used to reduce the friction of using the federated social network, e.g. Mastodon social network. Federated social network, which is a decentralized network, consists of many communities with servers run by different people or organizations. To join a community, a user needs to sign up with username and password because the login credentials are not portable among the servers. A public digital identity will solve this issue with SSO while at the same time protects the privacy of the users. This will make the federated social network much easier to use and more appealing to users.

7.3. Decentralized Website and Wiki

Wiki applications (e.g. Wikipedia-like web applications) and websites can use the decentralized database to store data. The data normally consists of text, image and some small video and audio files. The database is particularly suitable for text data, and should be able to handle small image files less than 3KB in size. It can support bigger files if the files are divided into smaller chunks. However, like most regular databases, the decentralized database is not the best place to store large multimedia files. If it is possible, these files should be stored on the file systems on servers or cloud storage, for example IPFS. The database should store the hashes of these files to make sure they are not tampered with.

The server for the website and wiki applications can be run locally with the full node as a decentralized application. For mobile applications or for whoever does not want to run a full node, they can connect to the Electrum-like peer-to-peer network as described in section 7.1. It is also possible for anyone to run a third-party server to provide the services.

8. Conclusion

We present a decentralized key-value database based on Bitcoin's proven distributed ledger and consensus protocol. The database is consistent across the peer-to-peer network, and the data is cryptographically tamper-proof and can only be updated by the owners who have the private keys. All the database operations can be accessed through JSON-RPC API, and no programming

of smart contract is required. Developing decentralized applications on this database is similar to developing on traditional databases. Its ease-of-use and security should provide much needed incentives for the development of more decentralized applications.

9. References

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System"

[2] Namecoin, https://namecoin.org

[3] H Kalodner, M Carlsten, P Ellenbogen, J Bonneau, A Narayanan, “An empirical study of Namecoin and lessons for decentralized namespace design”

[4] Maximilian Wöhrer and Uwe Zdun, “Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity”

[5] P Tsankov, A Dan, D Drachsler-Cohen et al., “Securify: Practical Security Analysis of Smart Contracts”

[6] IPFS Documentation - Pinning, https://docs.ipfs.io/guides/concepts/pinning/

[7] W3C, Decentralized Identifier Specification v1.0, Decentralized Identifier Specification v1.0

[8] W3C, DID Method Registry, https://w3c-ccg.github.io/did-method-registry/

[9] W3C, BTCR DID Method, https://w3c-ccg.github.io/didm-btcr/

[10] W3C, Use Cases and Requirements for Decentralized Identifiers https://www.w3.org/TR/did-use-cases/

[11] Electrum protocol specification, https://electrum.readthedocs.io/en/latest/protocol.html

[12] IPFS Documentation, https://docs.ipfs.io/