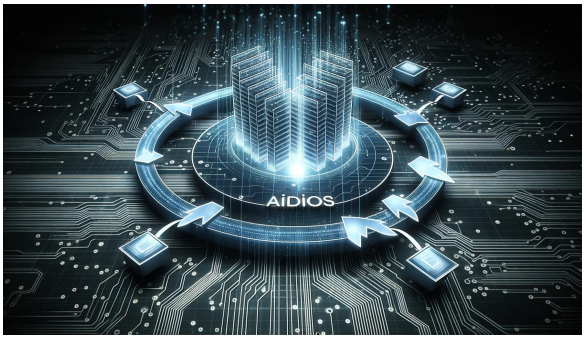
AIDIOS: Acyclic Immutable Decentralised Information Optimised Storage
Author: Jamie Gilchrist
Affiliation: Developer of AIDIOS
Contact: admin@aidios.io
Version: 0.2.1
Date: 01/09/23



Abstract
AIDIOS (Acyclic Immutable Decentralised Information Optimised Storage) is a novel decentralised storage protocol(complete with API), designed to store & access versioned data on existing blockchain networks. AIDIOS offers a fault-tolerant method for storing and retrieving versioned data on blockchain networks like Bitcoin, Bitcoin Cash, Litecoin and more. These options offer great flexibility for developers when choosing the most suitable storage solution, based on cost, availability and overall immutability. For example, BTC is widely accepted as the most established, decentralised and trusted network protocol. AIDIOS is efficient, inherently secure, verifiable, structured and all indexing/versioning is handled on-chain, with no need for external databases.

Background
On networks like Bitcoin, it is possible to store small pieces of data directly within a transaction. This can be done by crafting a raw transaction and leveraging a special field known as ‘op_return’. However, although this is a useful feature, it is limited in the fact that for networks like Bitcoin, the data has to be less than 80 Bytes, and Bitcoin Cash, slightly more at 220 Bytes. This small amount of (unversioned, unindexed) data is often not enough for real contextual meaning(as would be the case for NFT metadata, or other structured json). Often there is only room for a single hash. AIDIOS removes these limitations, and offers a solution which allows for any data size to be stored. This data(tied to the supplied public key) is versioned, and eternally available. On networks like Bitcoin Cash, the cost of storage is around £0.04 per KB, and free to retrieve forever.

Table of Contents

1. Introduction

Traditional centralised storage solutions are effective(especially for large amounts of data), however they lack certain properties which would make them suited to high importance, trustless data. AIDIOS proposes a radical shift from these conventional methods, by introducing a decentralised storage protocol, that leverages established blockchains to offer a wide array of benefits, such as, immutability, one time process(no running) costs and globally verifiable data.

2. Objectives

Security

Produce an inherently secure and redundant storage protocol, which leverages established blockchain protocols, to create truly decentralised and ‘forever’ storage.

Cost-Effectiveness

To provide an effective solution to store critical(or high importance, niche use case) data, in a cost efficient manner.

Efficiency

To create a protocol and hierarchical structure which allows for rapid, organised, fault-tolerant decentralised storage and retrieval of data.

Flexibility

To develop a fully fleshed storage protocol & framework to enable developers who work with multiple technologies to integrate them with AIDIOS seamlessly.

Simplicity
To abstract away blockchain complexities from the developer, offering a standardised API and SDK.

Restrictions
To allow for more structured, versioned arbitrary data to be stored on blockchain networks, thus removing the normal associated restrictions and complexities when dealing with op_return in isolation.

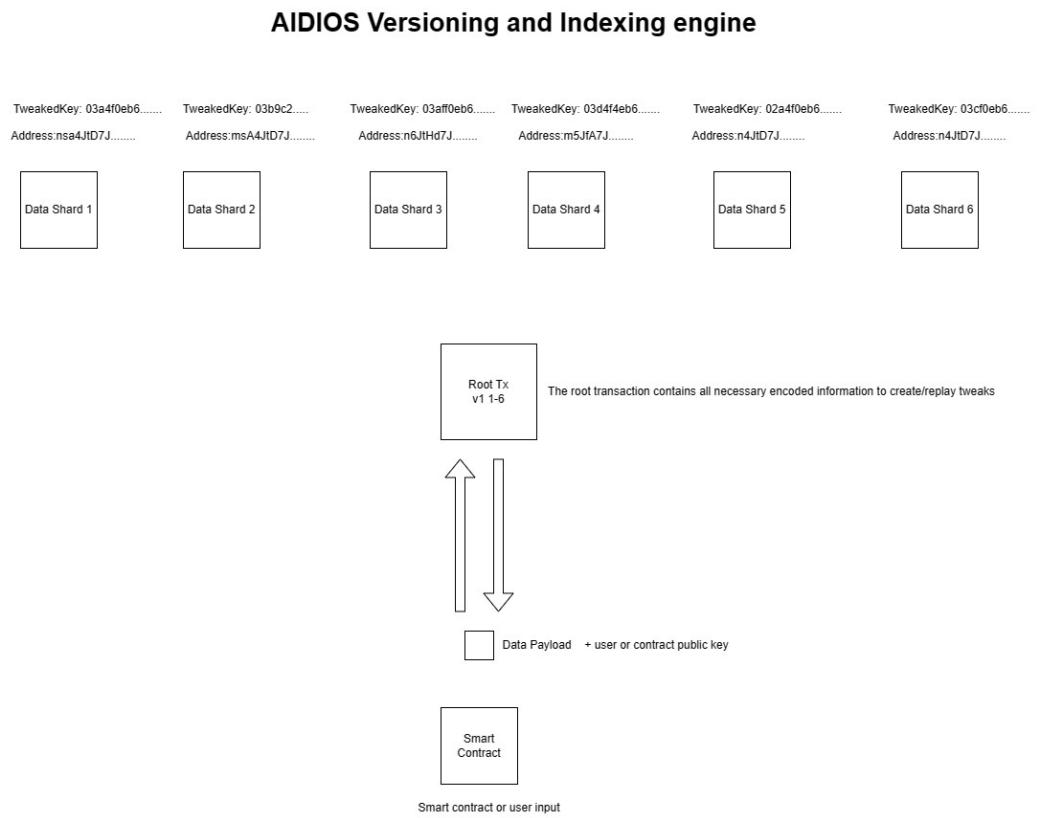
3. Technical Overview

3.1 KeyWeave

The backbone of AIDIOS is a data structure protocol called KeyWeave. KeyWeave allows for the efficient storage/retrieval of versioned data across multiple transactions. The structure ensures both data integrity and efficient data retrieval. KeyWeave works by tweaking an original(supplied) public key in such a way as to encode both versioning and indexing information about the data. This system allows for scalable storage/retrieval of data across many parts. As an additional feature, the addresses which are created by AIDIOS, are fully owned by the original key pair holder. We achieve this without having access to any sensitive information, and our steps can be repeated on the users private key(locally and offline) to generate the new keys for these addresses.

3.1.1 KeyWeave Example

The following illustration shows what the KeyWeave system looks like at a high level. Each of the components(nodes) are transactions. Each data part(or shard) has a dedicated Public Key, Public Address and a transaction, with a singular root node which encapsulates all the information required to run the sequence of tweaks needed to index the entire data set. One of the unique properties of this system is that each of these addresses is owned by the original key holder. I.e the Public key supplied: The user can apply the same tweaks as we have to their private key, enabling them to spend from the address, even though AIDIOS was never given any sensitive information. In the example below, we show a data structure containing 6 data shards, and a root transaction.



Note: without supplying the public key, finding the root node alone would not allow for the rebuilding of the data, by design. This is because the tweaking information needs to be applied to the original public key which was used to store the data.

3.2. Tweaked Public Key:

A tweaked public key is derived from an original public key and some additional data (called a "tweak"). The purpose of tweaking a public key is to create a new public key that is related to the original, but appears random to anyone who doesn't know the tweak. AIDIOS uses this as a foundation for including various metadata surrounding the record.

The tweaked public key P' is calculated as follows:

$$P' = P + H(P \,||\, m) * G$$

where:

- P is the original public key.
- m is the message or additional data.
- H is a cryptographic hash function (e.g., SHA-256).
- G is the generator point of the elliptic curve.
- || denotes concatenation.

AIDIOS Tweaking process:

Key Tweaking:

- Involves altering a base public key using additional data (like a version number and identifier) to produce a unique, derived public key, and subsequently, a Bitcoin address.
- Example: Base public key 3021f… is tweaked to produce a unique new key 023d.. And subsequent address mzAfr2..

Versioning and Indexing of Data Parts:

- Each file version is stored with a unique identifier.
- A file is split into several parts, and each part is indexed.
- Example: File Version 1 is split into 3 parts, indexed(expressed in simple form) as 1_1, 1_2, 1_3.

Root Transaction and OP_RETURN Data:

- The root transaction contains metadata about file versions and their parts.
- OP_RETURN data in the root transaction is a condensed representation of all parts and versions.
- For multiple versions and parts, the OP_RETURN data encapsulates this complexity in a concise format.

Example: Root Transaction for 2 Versions with 6 Parts

Let's consider a scenario where we have 2 versions of a file, each version having 3 parts.

Versions and Parts:

- Version 1: Parts 1_1, 1_2, 1_3
- Version 2: Parts 2_1, 2_2, 2_3

Root Transaction OP_RETURN Structure:

- The OP_RETURN data includes identifiers for each part of each version.
- It uses a compact format to list versions and their respective parts.
- Structure: V1:P[1-3];V2:P[1-3]
    - V1 and V2 denote versions 1 and 2.
    - P[1-3] indicates parts 1 to 3 for each version.
- This structure efficiently encapsulates information about multiple versions and parts, keeping the OP_RETURN data concise, allowing for scaling with the number of parts.

Potential for Unlimited Parts and Versions:

- The design allows for representing an extensive number of parts and versions.
- By using ranges and compact identifiers, the system can scale to handle many revisions and parts without significantly increasing the size of the OP_RETURN data.
- For instance, V1:P[1-100];V2:P[1-100]... can represent hundreds of parts across multiple versions within a manageable size.

Data Retrieval Process

Fetching Data:

- To retrieve a specific version, AIDIOS reads the root transaction's OP_RETURN data.
- It then identifies the parts associated with the desired version.

Reconstructing the File:

- Each part is fetched from its respective transaction on the blockchain.
- The parts are reassembled in the correct order to reconstruct the full file for the specified version.

Security and Efficiency

- Immutability: Blockchain transactions ensure that once data is stored, it cannot be altered, guaranteeing integrity.
- Privacy: Each part and version is associated with a unique, tweaked address, enhancing privacy and security.
- Efficiency: The condensed representation of parts and versions in the root transaction allows for efficient storage and retrieval, even as the number of versions and parts grows.
- Data can (optionally) be encrypted at source, or via a built in encryption provided by the API.

4. Status Bits

AIDIOS employs an intelligent bit-vector, referred to as Status Bits, to encode important metadata. This bit-vector is a compact way to include critical information about the data, such as its type, part number, and checksum.

```
"STATUS_BITS = {
'version': (0, 8), # The version of the of schema used(currently v1.0.0)
( 256 places)
'node_type': (8, 10), # Data and root.
'total_parts': (23, 31), #parts file is sharded into
'part_number': (31, 39),# tells the position of the fragment
'checksum': (39, 71), # data integrity each node and it's children.

}"
```

4.1: Status Bits Encoding

The Status Bits are encoded in such a way that they provide a compact yet comprehensive representation of the metadata associated with each data part.

Let S be the Status Bits vector, and $S_{version}, S_{node_type}, S_{file_type}, \ldots$ be its components.
The Status Bits are encoded as:

$$S = [S_{version}, S_{node_type}, S_{file_type}, \ldots]$$

The decoding function D extracts this information:

$$D(S) \rightarrow \{"version" : x, "node_type" : y, \ldots\}$$

Since each bit-vector component ***S*** is uniquely mapped to a metadata attribute, the Status Bits encoding is both compact and lossless.

4.2 Checksums and Data Integrity

AIDIOS employs SHA-256 for checksums for each data shard but uses only the first 32 bits. The choice for 32 bits is a trade-off between the level of data integrity assurance and the storage space required.

4.2.1: Checksum Integrity

The 32-bit checksum provides a sufficient level of data integrity while optimising for storage space. Given that SHA-256 is a cryptographic hash function with a near-zero probability of collision, truncating it to 32 bits still provides a high level of assurance against data corruption.

5. API Endpoints

The AIDIOS API provides several endpoints that facilitate easy interaction with the protocol. They are designed with REST principles in mind, offering intuitive methods for storing and retrieving data. The API documentation provides examples in multiple programming languages, from Python to Java and Go.

Store a versioned file:

```
 curl -X POST    -F "file=@file.txt"    -F
"original_pubkey=032c0022708805631615842fda711e37b6c41199fbd19a11571f3f4
41dc02c0cc6"    -F "encrypt=no"      https://apiv2.aidios.io/store
 {"message":"File stored successfully with root transaction
ID.","root_txid":"a1c688376f244b093847ecaf31aa268be6eff426a6a693e38e12e7
361b4635d4","status":"success"}
```

Retrieve by version:

```
curl -X GET
"https://apiv2.aidios.io/retrieveversion?publicKey=032c00227088056316158
42fda711e37b6c41199fbd19a11571f3f441dc02c0cc6&message=1"

{"content":"SmFtaWUgbmV3IHRlc3QgZmlsZQo=","message":"File retrieved
```

```
successfully","status":"success"}
```

Retrieve by root txid:

```
curl
"https://apiv2.aidios.io/retrieve?root_txid=94e5c968b559203140e396ed6b4e
e5b46ba84202509673a0c3ae6d5f1ee1734f&original_pubkey=032c002270880563161
5842fda711e37b6c41199fbd19a11571f3f441dc02c0cc6"
```

Check total versions by Public Key:

```
curl -X GET
"https://apiv2.aidios.io/versions?original_pubkey=032c002270880563161584
2fda711e37b6c41199fbd19a11571f3f441dc02c0cc6"
{"status":"success","total_versions":1}
```

6. Use Cases

ADIOS has a wide range of use cases when considering high-impact applications which can leverage, independently verifiable, immutable storage:

- Healthcare records
- Educational records
- Financial data
- Land registration
- Decentralised voting
- Content Management
- Decentralised Identity Documents
- Auditible Environmental control
- Power system data
- Data Governance
- & many more applications

7. Cost comparisons

Direct cost comparisons become slightly difficult when considering AIDIOS. There is no other solution which allows for versioned storage of data(beyond op_return limits) on existing blockchain networks. AIDIOS can run in a dedicated ‘Store all data on chain’ mode, or for providing verbose metadata storage(where the data is stored elsewhere, such as IPFS, for

example.

7.1 Comparing equivalent storage in Ethereum V Bitcoin Cash

A basic cost comparison

Ethereum Storage Costs:

- 500 bytes: $393.75
- 1 KB: $806.40
- 2 KB: $1612.80

ADIOS (Bitcoin Cash) Storage Costs:

- 500 bytes: $0.02 (approximately 2 cents)
- 1 KB: $0.04
- 2 KB: $0.07

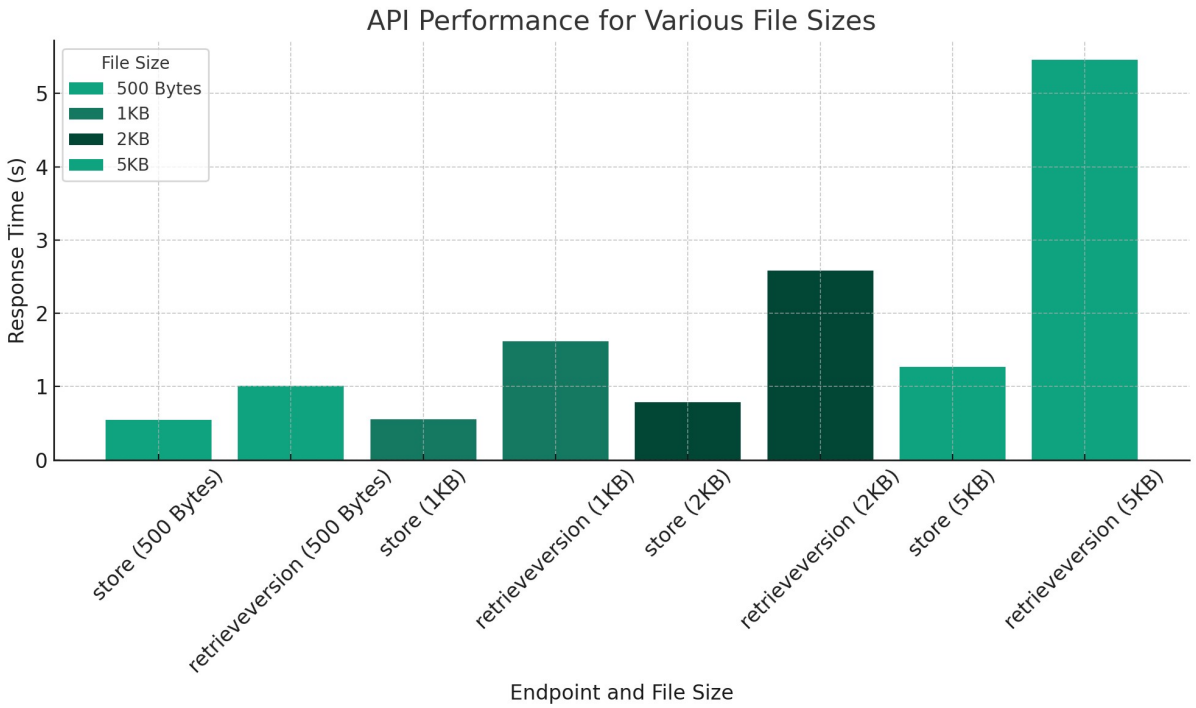
    - In each of the above, AIDIOS(when combined with Bitcoin Cash) is over 13000X more cost effective than Ethereum.

8. Performance:

In regards to performance, AIDIOS presents a unique combination of efficiency, inherent security scalability, and cost-effectiveness. When compared to classical storage systems(SAN, NAS, for example), the performance results might seem lacklustre. However the comparison between traditional storage, and the immutable(and independently verifiable, highly available) nature of data stored with AIDIOS is difficult to directly compare against.Traditional storage solutions prioritise speed and capacity, often at the expense of security and verifiability. AIDIOS, in contrast, offers fully immutable data storage that is not only independently verifiable, resistant to tampering, and one-time cost, but also guarantees high availability.

Below are the performance results for AIDIOS. In this test we assume:

- 4X Tests, 500 Bytes, 1KB, and 2KB and 5KB
- Response times are in milliseconds and seconds(where relevant).
- Each test was conducted 10 times to determine a mean value for each.
- Concurrency tests were also done(up to 10X, with no noticeable change in response times)
- AIDIOS version 0.2.1 is tested
- AIDIOS is hosted on a single (small) EC2 instance for testing.



9. Security Measures

AIDIOS employs a multi-layered security architecture that includes access control . AIDIOS treats input data as binary, and so developers are free to encrypt data at source. Much of the underlying security is inherited by the blockchain design AIDIOS leverages. Simply put, Bitcoin is a secure protocol, which is tried and tested.

10. Future Work

Future developments for AIDIOS include the integration of more advanced cryptographic techniques, the development of further SDK's and support for multiple chains.
Planned performance improvements aim to further reduce response times across the board.

11. Conclusion

By leveraging innovative data structures backed by solid cryptographic techniques, AIDIOS offers an immutable, scalable, and cost-effective solution for storing high-impact data. The protocol's versatility makes it suitable for a wide range of applications, with the potential to have a transformative impact on various sectors. In terms of cost-efficiency alone, it poses a real value-add for developers looking to integrate applications leveraging specialised storage.

12. Glossary

AIDIOS (Acyclic Immutable Decentralised Information Optimised Storage):A storage protocol, designed for efficient, scalable, fault tolerant storage and management of records on existing blockchain networks.

Blockchain:A peer-to-peer digital network, with a decentralised and distributed ledger.

op_return: An opcode used in Bitcoin-like to mark transaction outputs as 'provably unspendable', then allowing for the addition of arbitrary data.
KeyWeave: A proprietary I data structure protocol at the core of AIDIOS, inspired by Merkle trees and Directed Acyclic Graph (DAG) technology, allowing for efficient and decentralised storage.
Directed Acyclic Graph (DAG): A graph with no directed cycles, commonly used in data processing, scheduling, and/or data storage solutions.
SHA-256: An industry standard cryptographic hash function producing a 256-bit (32-byte) hash value, most commonly represented as a base64 representation.
Status Bits: An intelligent bit-vector developed for AIDIOS to encode essential metadata about the data being stored.
UTXO (Unspent Transaction Output): The output of a blockchain transaction(earlier) that has not been spent and can be used as an input in a new transaction(current).
API (Application Programming Interface): An interface offering protocols and tools, allowing multiple programming languages to communicate with a given software.
SDK (Software Development Kit): A set of software tools with libraries that developers use to create applications for specific platforms.
SAN (Storage Area Network): A fast network providing access to centralised, block-level data storage.
NAS (Network-Attached Storage):A multi protocol file sharing server, allowing multiple clients to connect to store and retrieve information, typically as a logically mounted volume.
Decentralised Storage: A method where data is stored across a network of decentralised nodes rather than centralised servers or systems. This is different from 'distributed' storage, and the concepts are unique to blockchain.
Bitcoin Cash (BCH): A fork of Bitcoin. It increases the op_return size to 220Bytes, and increases the block size (to 32MB), allowing more data to be stored and transactions to be processed.
Ethereum: An alternative open-source, blockchain-based platform featuring smart contract(turing complete) functionality.
Node: A point of intersection/connection within a network, typically representing a device or system, or other construct.

13. References

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.
Merkle, R. C. (1987). A Digital Signature Based on a Conventional Encryption Function.
Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.).Introduction to Algorithms, Third Edition (edutechlearners.com)

14. Appendix

- API documentation
- SDK documentation

15. Acknowledgments

I would like to thank Hal Finney for creating the first 'reusable proof of work', Satoshi for bringing Bitcoin to the world, and to all the other open source contributors since, who have

then worked tirelessly to help realise the full potential of blockchain and other Distributed Ledger Technologies.