From Paper to Blockchain: A Proof of Concept for Storing Aviation Mainentance Documents

Documentation management

- Every item on the market requires accompanying approved documentation.
- The approval process (P) involves reviews by subjects Y and Z.
- Approved documentation should be published and maintained for a specified period (T).
- Any updates to the product necessitate corresponding documentation updates, subject to approval following the set procedure.

Further requirements

- Documentation Versioning: It is important to keep track of all documentation versions for each product
- Documentation Accessibility: Documentation should be easily accessible to all stakeholders, including support teams, sales, engineers, etc.
- Documentation standardization: To make it easier to create and read documentation, it should follow some templates or standard guidelines.
- Confidentiality and Data Protection: Depending on the nature of the product and the industry, it should be possible to protect sensitive information contained in the documentation.

“Classically” how does X guarantee these properties? Through checks, which verify that all these things are respected. However, the controls are expensive and not very effective:

1. For example, a check verifies that the file is available at time t, but then the file may become unavailable
2. Given the cost, checks can be done on a random basis
3. Do not provide evidence to all stakeholders that requirements have been met (i.e., controllers must be trusted)

Automatize, of course

- Automating via a Document Management System (DMS): enforces process requirements.
- Hosting Dilemma: Institution (needs IT skills & infrastructure) vs External Supplier (requires trust, potential lock-in).
- Interoperability Concerns: Deciding on open APIs and their access permissions.

With smart contracts

```
contract DocumentManagement {
    struct Document { bytes32 hash; bool isApproved; }
    mapping(uint => Document) public documents;
    mapping(address => bool) public controllers;

    function submitDocument(bytes32 _hash) public {
        // ...
        // add document to 'documents' with msg.sender as
producer
        // set isApproved to false
        // ...
    }

    function approveDocument(uint _id) public {
        // ...
        // check if msg.sender is a controller
        // set isApproved to true for document with id _id
        // ...
    }

    function addController(address _controller) public {
        // ...
        // add _controller to 'controllers'
        // ...
    }

    function isApproved(uint _id) public view returns(bool) {
        // ...
        // return approval status of document with id _id
        // ...
    }
}
```

Problems

This opens up some problems:

- How does the on-chain component ensure that documents are available?
- How do we manage access control to read data?

Checking data availability

How can it be verified that the documents that the various actors (for example the proposer) said they have made available, by committing on their hashes, are indeed available?

- The proposer could publish a hash, but then keeping from making the whole document available
- This is the same problem you would have trying for instance to implement a registered mail with return receipt

Possible solution: Filecoin

- Filecoin and IPFS: Pay providers to ensure document availability.
- Workflow smart contract: Document proposer proves payment for Filecoin storage.
- Verification methods:
    - Oracle: Reliable but potential centralized point of failure.
    - Crosschain communication: Enhanced decentralization but adds complexity.
- Potential Challenges:
    - Ensuring authenticity and timeliness of proofs.
    - Managing costs associated with data storage and retrieval.

Oracles

- Use existing solutions like Chainlink, Kleros.
  - Costly for frequent checks.
- Solution: Incentive/Punishment Mechanism:
  - Documentation publishers stake for document availability.
  - Challengers can invoke an oracle if file isn't accessible.
  - If file unavailability is verified, stake is forfeited.
- Potential Issue:
  - Publishers might publish after being challenged.

Ethreum data blobs

Problem:

- Assure data availability for Ethereum rollup transactions.
- Balance data size and verification effort

Solution: Data Availability Proofs (DAPs):

- Publisher commits to a file, F, via a cryptographic.
- Erasure Codes:
    - The file F is encoded into n blocks using Reed-Solomon codes.
    - Any subset of k blocks ($k \leq n$) can reconstruct F.

Checking Strategy:

- Probabilistically check m blocks ($m \leq n$).
- With each successful check, the probability of insufficient data availability decreseas exponentially

- Upcoming hard fork likely to implement data availability.
  - This introduces cheap, provably available block space, known as 'data blobs'.
- Potential Usage:
  - Could these data blobs be used for storing arbitrary documentation, such as regulatory or product data?
- Long-term Availability:
  - Despite the promising use cases, Ethereum doesn't offer a long-term solution.
  - The data blobs' availability isn't guaranteed indefinitely, presenting potential future data access issues.
- Looking Forward:
  - Exploration of alternative storage solutions or extended availability guarantees is needed.
  - Potential integration with other decentralized storage systems could enhance the long-term availability and reliability of data.