LegitDoc: A platform for decentralized verification of digital documents and transactions.

Sheshavishnuprasad Dharanaiah
CTO, Zupple Labs Pvt. Ltd.
shesha@legitdoc.com

Neil Kevin Martis
CEO, Zupple Labs Pvt. Ltd.
neil@legitdoc.com

Rishab Verma
CPO, Zupple Labs Pvt. Ltd.
rishab@legitdoc.com

November 2022

Abstract

Official records, in the form of physical and/or digital documents, function as an attestation of a certain fact, agnostic to respective industries – documents such as land records, education certificates, driving licenses, etc.; are often commonly recognizable examples. Such official records act as a proof of a given transaction – transactions which are packaged in the form of publicly verifiable documents, issued to respective stakeholders by the central entities maintaining the records.

For stakeholders, wishing to verify such documents, ensuring authenticity of official records via a tamper-proof process is fundamental. Today, several technologies strive to improve the process of document verification, both for physical and digital documents, via various centralized solutions. Although, many centralized solutions have raised the bar for tamper proof verification via digitization, requirements for customer ease, tamperproof verification, and process efficienciencies still exist. Utilizing examples of centralized system failures, even major entities such as Aadhar, Facebook, Twitter, etc. continue to showcase failures[1] – due to the advancements in digital technologies as a whole and, the ever-present threat of rogue internal actors of a given central system.

Public Blockchain technology has enabled a new era for producing documents, which are instantly verifiable, private, and decentralized, effectively enabling 100% tamper-proof verification guarantees – combination of benefits which are not present in centralized solutions. This paper highlights how the LegitDoc platform thrives to become a standard interface for public document verification, along with solving critical decentralized application adoption challenges such as ease of use, application scalability and navigation around high transaction fees. LegitDoc's vision is to become a standard for the issuance and verification for documents, for industry agnostic

examples such as bank guarantee, citizen certificates, consent document, education certificate, loan document, hospital invoice, or an airline invoice, etc. To achieve this, LegitDoc leverages security of public blockchains, to issue documents that are secure, private, non-repudiable, and tamper-proof. The technology behind LegitDoc makes the process of document authentication and verification, highly inexpensive while providing a flexible path towards user adoption.

1 Motivation

Globally, documents are produced in substantial volumes – with an objective to credentialize official records and transactions. Document authentication and ease of verification, however, remains to be a challenge for appropriate stakeholders. To overcome these challenges, a platform must enable instant document issuance and verification, while preserving security and privacy of the stakeholders, and more importantly, enabling transparency in the verification process.

Today, centralized solutions, attempting to produce verifiable documents, exist in the market. These solutions, however, give up on stakeholder security and privacy, and verification process transparency. Centralized solutions continue to possess fundamental flaws for the end-to-end document issuance and verification process – specifically, such solutions utilize vulnerable centralized authorities (CA), which act as digital signature/key validators for the entire solution process, opening an opportunity for malicious behavior and increasing system attack vectors. Therefore, compromising stakeholder requirements such as non-repudiations, document back dating prevention and document invalidation. In addition to the systemic CA risk, in centralized solutions, process transparency for public stakeholder is often nonexistent, and verification processes are tedious and expensive.

1.1. Centralized Digital Signatures

Facility for centralized digital signatures on documents like pdf had solved the problem of verifiable documents to an extent. There are following major problems with centralized digital signatures:

(a) **Non-Repudiation**
Once the issuer has issued a document, he must not be able to be dishonest about the issuance. Meaning, it must not be possible for the issuer to erase/alter the history of an issuance. Giving the document holder or verifier the confidence about non-repudiation of the document is something that can be easily achieved through blockchain, and has never been made possible through centralized systems. When using just digital signatures, such history of issuance can be influenced by the issuer or the CA.

(b) **Trusting the CA**
The public keys used for digital signatures need to be authentic, and Certificate Authorities have been used for a long time now as a trusted third party to validate if the public key indeed belongs to the issuer. This process of authenticating the issuer through the CA has been completely opaque, and even if we let go of the possibility of third-party transgression, there has been several instances of the CA getting hacked that has led to some major attacks[2][3].

(c) **Backdating**
Backdating means claiming to have created a document at an earlier date than the actual one. Centralized digital signatures solve this by having the trusted third parties to timestamp the digital signature on the document. By what we know so far, third parties haven't been as effective in protecting themselves from external or internal malevolence.

(d) **Document Invalidation**
Once the document is signed and issued to the document holder, there is no way that the issuer can invalidate such a signed document so as to indicate the verifier that the issuer no longer vouch for the document. Although this can be facilitated by CAs providing a custom feature to do so, it boils down to "Trusting the CA".

(e) **User Experience**
There have been several versions of PDF readers, and they carry out the signature verification in different ways. The digital signatures have sometimes been incompatible with newer version of the PDF readers. Recently, an attack called "Shadow Attack"[4] displayed the importance of User Experience and how it was exploited to create fake documents that were thoroughly indicated as valid at the verifier's end. Also, a recent study [5] had shown that majority of pdf readers were vulnerable to at least one kind of attack.

2 Introduction

Public blockchains, Ethereum and Polygon as example, provide transparent mechanisms for instantaneous verification of data which is stored on chain. With the increase in public blockchain accessibility, such distributed ledgers provide a general-purpose way to represent ownership of digital assets such as documents, that maintain their authenticity, history, and ownership state across the web. While the magnitude of decentralized applications and their use-cases continue to increase, instant and transparent data verifiability continues to be one of ideal use-cases. We aspire to leverage public blockchain's superiority of tamper proof data verifiability – a claim backed[6][7] by industry leaders such as Balaji Srinivasan (former CTO of Coinbase)

and Joshua Ledbetter, along with our very own thesis[8], to solve the instant document authentication and tamper proof verifiability challenge.

As with the state of decentralized applications, and the stakeholder requirements for the instant document authentication and tamper proof verifiability challenge, there do exist 3 primary challenges that we had to overcome, to build a standard interface for public document verification. Namely, these obstacles are as follows:

2.1. Ease of Use - Customer Adoption

Public blockchain interaction and decentralized application adoption continues to be poor, a feat resultant primarily due to a need of technical prerequisites and, difficult to adopt application user interfaces. While few in the web3/decentralized application industry are technologically equipped to manage their own digital identities and private keys (through wallets such as Metamask), majority population continues to rely upon centralized application bridges when accessing (conducting transaction on) public blockchains. Examples of centralized exchanges having larger customer adoption, with the existences of decentralized exchanges (for e.g., Uniswap), is a prime indication that customers prefer refraining from ownership of technological overheads, such as the above-mentioned digital identity and private key management.

Similarly, when it comes to document verification via public blockchains, to drive user adoption, both the issuer and verifier need a simple convenient to use interface, that abstracts out the technological overheads that are present when interacting with public blockchain - an interface preserves preserve the property of tamper-proof authentication and verification. In the upcoming section 3a, we lay out how the LegitDoc platform solves this user adoption challenge utilizing Multisignature schemes.

2.2. Transaction throughput - Application scalability

Businesses of all sizes, whether public or private, and agnostic to their industry of function, tend to generate large volumes of official records on digital and physical documents. Such stakeholders prefer the lowest latencies for their application transactions. For application transactions such as document issuance and verification, a near real time application transaction process speed is preferable.
While certain public blockchain promise to address high throughput requirements, in the event of congestion, or service disruption[9], latency guarantees simply can't be delivered. In addition to low latency guarantees and its downstream effects on application throughput, nonce management[10] is another obstacle that drastically limits the throughput of the address transacting with a smart contract. Even when performance optimizations like batching the transactions from individual issuers can be done, the time to finality is not guaranteed when multiple issuers are competing to commit the transactions.

2.3. High blockchain-anchoring cost – Storage fee

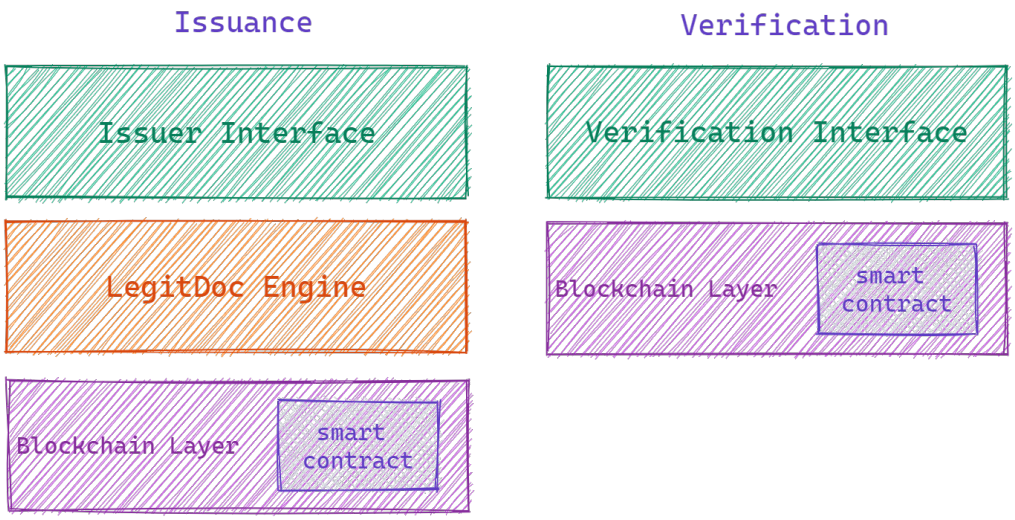
With the volatile cost of public blockchain tokens, the cost to store data types such as document hash, digital fingerprint etc., on public blockchains, even with data sizes as small as 32 bytes, is expensive and unpredictable. While it seems feasible to store the fingerprint/hash of each issued document on L1 chains, like Ethereum and L2 chains like Polygon, even with the unpredictable and high transactions costs, when the scale of transaction requirement reaches hundreds of millions of documents, a batching mechanism is required. Anchoring individual document hashes on public blockchain linearly is neither scalable nor efficient.

3 LegitDoc System

The LegitDoc System facilitates issuing and verification of documents using Blockchain technology. It achieves this using the following modules:

(a) Issuer Interface - Web Application or SDK
(b) LegitDoc Engine - An API-first REST application
(c) LegitDoc Smart Contract
(d) Verification Interface - Web Application

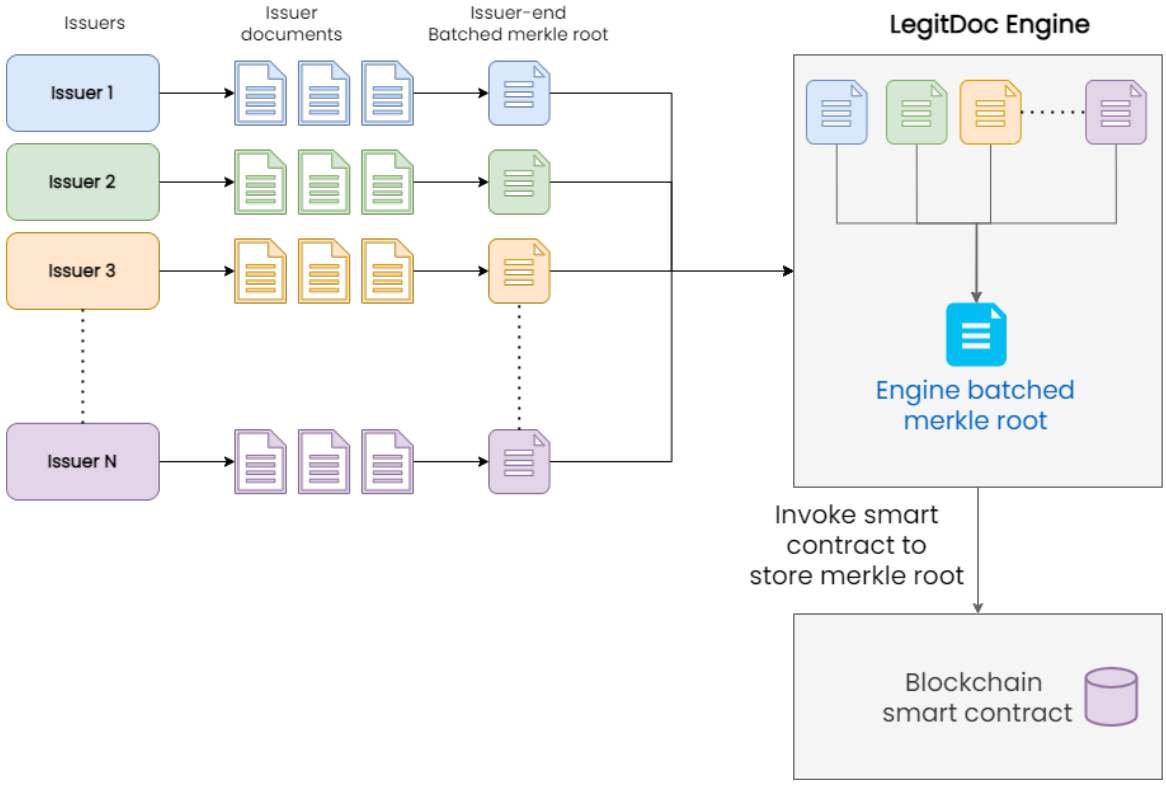
Figure 1: LegitDoc Stack



During issuance, the issuer interface communicates with the LegitDoc's underlying Engine, which in turn communicates with smart contract on public blockchains, namely Ethereum and Polygon. This specific middle layer, ensures the issuer is abstracted out from transacting with the blockchain layer and the intricacies that come with it, therefore driving drive user adoption.

For transaction requests on the issuer interface (web application), which are ported via the LegitDoc Engine, each public blockchain transaction request is also signed by the issuer's private key. While LegitDoc owns the smart contract, the application layer ensures the issuer's intent is registered on the smart contract. On the other end of the spectrum, during verification, the verification interface (web application) communicates directly with the blockchain layer, where the verification interface is open-sourced – open for tamperproof verification and authentication without the need for a third party.

Figure 2: High-Level view of the LegitDoc System



3.1. Issuer Registration

Each issuer who wants to issue documents on LegitDoc can sign up with the LegitDoc portal. During registration, the issuer can use a hardware wallet or generate a digital signature key pair using the LegitDoc Sign browser extension. The issuer is responsible to keep the private key secret and safe. Only the public key is shared with LegitDoc which completes the issuer registration. LegitDoc Smart Contract is invoked to register this issuer on the public blockchain. A unique id (auto-generated), name, and public key tuple is registered on the smart contract storage.
The authenticity of an issuer is important, and the smart contract facilitates decentralized vouching to vouch for the authenticity of the issuer. Trustworthy organizations

or DAOs can be a witness and vouch for an issuer after a manual verification process. Having enough witnesses elevates the issuer to a "verified issuer" status on the smart contract. See the last sections of the paper for more on Rolling private keys and recovering lost issuer keys.

3.2. Issuance

The Issuer interface is a REST API SDK or a web application that implements this SDK. The Issuer interface talks to the LegitDoc Engine to submit an anchoring/issue request. The LegitDoc Engine responds with a confirmation to such a request with bounded time guarantees.

An issuer can opt to issue documents one after another or in batches. A Merkle Tree[11] of all the documents in the batch is formed to compute the Merkle Root MR_I. Each leaf L of this Merkle Tree corresponds to each document in the batch, and is computed as follows:

$$\begin{aligned} D_I &= H(d) + E_d + E_i \\ S_I &= S(H(D_I)) \\ L &= H(S_I) \end{aligned} \tag{1}$$

where d is the byte data of the digital document to be issued. H is the SHA3[12] hash function. S is SECP256K1[13] sign function that uses the issuer's private key to sign the data. L is the leaf that's participating in the Merkle Tree for the corresponding document d. E_d is the expiry timestamp of the document set by the issuer after which the document is no longer valid. E_i is the invalidation expiry timestamp of the document set by the issuer after which the document can no longer be invalidated explicitly by the issuer.
To sign the hash of D_I, the issuer can either load the private key in the SDK, or in case of using a web interface, load it onto the LegitDoc Sign browser extension, or use a hardware wallet.

The MR_I computed out of such leaves is then submitted to the LegitDoc Engine. The Engine responds with a batch id that MR_I is now a part of. This batch id is an identifier of the batch that's currently running on the Engine. Like how the Issuer interface batches multiple documents, the Engine also batches multiple MR_I received from different issuers. A Merkle Tree is formed for each batch to compute the ultimate Merkle Root MR_U that's registered on the smart contract storage. The leaves of this Merkle Tree are the MR_I of each request received from issuers in the batch ongoing in the engine. The batching is done at fixed time intervals at the engine, which ensures that the engine talks significantly lesser with the public blockchain. This enables LegitDoc to anchor MR_U to multiple blockchains with cheaper costs.
The Issuer can poll to check for the status of the batch id received in the anchoring response. Once the batch is anchored onto the blockchain, the Engine reflects that

status in the response with other important data:

- txHash: The transaction hash in which MR_U was anchored on the blockchain.
- MP_U: Merkle Proof for MR_I's participation in computing the ultimate Merkle root MR_U

For each document at the issuer's end, the issuer interface generates verification data VD that is used to verify the authenticity of the document by any verifier. VD for each document consists of the following:

- txHash: The transaction hash in which MR_U was anchored on the blockchain.
- network: Name of the network on which the document is anchored.
- MP_U: Merkle Proof for MR_I's participation in computing the ultimate Merkle root MR_U
- MP_I: Merkle Proof for document d's participation in computing the intermediate Merkle root MR_I
- $IssuerID$: The unique ID of the issuer.
- MR_I: The intermediate Merkle root.
- E_d: Expiry timestamp on the document after which the document is considered expired.
- E_i: Invalidation expiry timestamp after which the document can no longer be invalidated by the issuer.
- S_I: The signature computed using issuer's private key as shown in Equation 1

The VD for each document is packed with the document in the form of a zip file, or if the original document is a pdf, the VD is embedded in the pdf. This zip file or pdf is ultimately dispatched by the issuer to the end consumer/document holder.

3.3. Verification

Verification can be done on the verification interface or on-chain. The verification interface is a web application that is open-sourced and facilitates decentralized verification of the issued documents. Issued digital documents consist of the original document data OD as well as the embedded verification data VD. The verification interface extracts the OD and VD out of the issued document. It follows the following procedure to verify the authenticity of the document:

1. Hash OD to get $H(d)$
2. Using E_d and E_i in VD and $H(d)$ from the above step, compute D_I as shown in Equation 1
3. Fetch issuer name and public key ($pubKey$) from the Blockchain using $IssuerID$.
4. Using $pubKey$, verify if S_I is a valid signature on D_I
5. Hash S_I to get $H(S_I)$
6. Compute MR_I using $H(S_I)$ and MP_I
7. Verify if computed MR_I is equal to MR_I in VD

8. Compute MR_U using MR_I and MP_U
9. Verify if MR_U is anchored on the blockchain
10. Verify if $H(d)$ is not invalidated on the blockchain - More on this later.

If the verification process can satisfy all the above steps, the document is a valid document issued by the $IssuerID$, and can be guaranteed to be tamper-proof.

Proof: MR_U cannot be computed without MR_I, and MR_I cannot be computed without the hash of signature on the document since $L = H(S(H(H(d) + E_d + E_i)))$. This means that the signature of the issuer on the document hash is mandatory to compute MR_U. This proves that assuming a collision-resistant hash function, MR_U cannot be computed arbitrarily to tamper with the verification process.

3.4. Invalidation

For several issuers, document invalidation is a critical use case. But, unfortunately, unlike issuance that can be batched, invalidation cannot be batched. Since the issued document is already dispatched to the document holder, the issuer has no control over the document once issued. The only sensible thing the issuer can do is to record on the blockchain that he intends to invalidate a particular document hash so that the verification process can check if the document hash is still valid.

During invalidation, Original Document data d is hashed and signed to get $S(H(d))$. On the blockchain smart contract, a map data-structure is used to map $H(IssuerID + d)$ to $S(H(d))$. It's important to capture the signature of the issuer because any third party including developers of LegitDoc should **not** be able to impersonate an issuer's invalidation.

As it can be imagined, the invalidation can't be as cheap as issuance, and the issuer needs to pay significantly more for each invalidation.

Merkle Root Invalidation: After a batch of documents has been issued, the issuer can invalidate the same batch by mapping $H(IssuerID + MR_I)$ to $S(MR_I)$ on the smart contract map.

Time based Invalidation: An issuer can also invalidate all documents by giving a time window. A document is considered to be invalidated if the issuer has recorded a particular time window as invalidated on the blockchain and the document was issued in that time window.

The Issuer interface facilitates the invalidation by talking to the LegitDoc Engine which talks to the blockchain layer.

In order to reduce the attack surface when an issuer's signature keys are compromised, Merkle root Invalidation and Time based Invalidation require LegitDoc's manual intervention as shown in the smart contract psuedo-code.

4 LegitDoc Smart Contract

Any blockchain platform that supports smart contracts can be used for facilitating LegitDoc's issuance, verification, and invalidation.

Pseudo code:

```
1
2  contract LegitDoc {
3    address[] workers;
4    address[] witnesses;
5
6    struct IssuerPubKey {
7        uint256 timestamp;
8        uint256 pub_key;
9    }
10
11   struct Issuer {
12       string id;
13       string name;
14       IssuerPubKey[] pub_keys;
15   }
16
17   mapping (string => Issuer) issuers; // Map Issuer ID to issuer
18   mapping (uint256 => uint256) roots_map; // Map Merkle root to time of
       issuance
19
20   struct Vouch {
21       address witness;
22       string issuer_id;
23       bytes signature;
24   }
25
26   mapping (string => Vouch[]) vouches;
27
28   struct Invalidation {
29       bytes signature;
30       uint256 timestamp;
31   }
32
33   struct TimeWindow {
34       uint256 start_time;
35       uint256 end_time;
36       Invalidation inv;
37   }
38
39
40
41   mapping (uint256 => Invalidation) single_invalidations;
42   mapping (uint256 => Invalidation) root_invalidations;
43   mapping (string => TimeWindow[]) time_invalidations;
44   uint min_worker_balance = 30;
```

```
45
46    // Anchor Merkle Root onto blockchain
47  putRoot(uint256 merkle_root) {
48    assert(workers array contains msg.sender);
49    assert(!roots_map[merkle_root]);
50    roots_map[merkle_root] = current_timestamp;
51    refill(msg.sender); // Refill with the fee being spent for this
   transaction;
52  }
53
54  // Invalidate a single document
55  invalidateSingle(uint256 key, bytes signature, string issuer_id) {
56      assert(workers array contains msg.sender);
57      assert(!single_invalidations[key]);
58      pub_keys = issuers[issuer_id].pub_keys;
59      assert(verifySignature(signature, key, pub_keys[pub_keys.length -
   1].pub_key));
60      single_invalidations[key] = Invalidation({
61          signature: signature,
62          timestamp: current_timestamp
63      });
64      refill(msg.sender);
65  }
66
67  // Invalidate entire batch of documents with their merkle root.
68  invalidateRoot(uint256 key, bytes signature) {
69      assert(owner == msg.sender);
70      assert(!root_invalidations[key]);
71      pub_keys = issuers[issuer_id].pub_keys;
72      assert(verifySignature(signature, key, pub_keys[pub_keys.length -
   1].pub_key));
73      root_invalidations[key] = Invalidation({
74          signature: signature,
75          timestamp: current_timestamp
76      });
77      refill(msg.sender);
78  }
79
80  // Invalidate documents issued within start_timestamp and end_timestamp
81  invalidateTimeWindow(string issuer_id,
82  uint256 start_timestamp,
83  uint256 end_timestamp,
84  bytes signature) {
85      assert(owner == msg.sender);
86      pub_keys = issuers[issuer_id].pub_keys;
87      assert(verifySignature(signature, start_timestamp+"_"+end_timestamp,
     pub_keys[pub_keys.length - 1].pub_key));
88      time_invalidations[issuer_id].append(TimeWindow({
89          start_time: start_timestamp,
90          end_time: end_timestamp,
91          inv: Invalidation({
92                  signature: signature,
```

```
                timestamp: current_timestamp
            })
    }));
    refill(msg.sender);
}

// Check if the document is invalidated on blockchain
isInvalidated(uint256 single_key, uint256 root_key, string issuer_id,
 uint256 inv_expiry_timestamp, uint256 issuance_timestamp) {
    inv = single_invalidations[single_key];
    if (inv && inv_expiry_timestamp <= inv.timestamp) {
        return {
            type: "single",
            timestamp: single_invalidations[single_key]
        }
    }
    inv = root_invalidations[root_key];
    if (inv && inv_expiry_timestamp <= inv.timestamp) {
        return {
            type: "root",
            timestamp: root_invalidations[root_key]
        }
    }


    for each timeWindow in time_invalidations[issuer_id] {
            if (issuance_timestamp >= timeWindow.start_time and
 issuance_timestamp <= timeWindow.end_time) {
                if(inv_expiry_timestamp <= timeWindow.inv.timestamp) {
                     return {
                        type: "time",
                        timeWindow: timeWindow
                    }
                }
            }
        }
    return "";
}

getIssuerDetails(string issuer_id) {
  return issuers[issuer_id];
}

getRootTimestamp(uint256 merkle_root) {
  return roots_map[merkle_root];
}

addWitness(address addr) {
  assert(owner == msg.sender);
  witnesses.add(addr);
  send();
}
```

```
143
144 addWorker(address addr) {
145   assert(owner == msg.sender);
146   workers.add(addr);
147   send_currency(addr, min_worker_balance);
148 }
149
150 // Vouch for an issuer
151 vouch(string issuer_id, bytes signature) {
152     assert(witnesses array contains msg.sender);
153     for each vouch in vouches[issuer_id] {
154         if(vouch.witness == msg.sender) {
155             abort;
156         }
157     }
158
159     vouches[issuer_id].append(Vouch({
160         witness: msg.sender,
161         issuer_id: issuer_id,
162         signature: signature
163     }));
164 }
165
166 getVouches(string issuer_id) {
167     return vouches[issuer_id];
168 }
169
170 registerIssuer(string issuer_id, uint256 pub_key, string name) {
171     assert(owner == msg.sender);
172     assert(!issuers[issuer_id]);
173     issuers[issuer_id] = Issuer({
174         id: issuer_id,
175         name: name,
176         pub_keys: [IssuerPubKey({
177             timestamp: current_timestamp,
178             pub_key: pub_key
179         })]
180     });
181 }
182
183 // Change the keys of an issuer
184 rollKey(string issuer_id, uint256 new_pub_key, bytes signature) {
185     assert(owner == msg.sender);
186     assert(issuers[issuer_id]);
187     pub_keys = issuers[issuer_id].pub_keys;
188     assert(verifySignature(signature, new_pub_key, pub_keys[pub_keys.
    length - 1].pub_key));
189     pub_keys.append(IssuerPubKey({
190         timestamp: current_timestamp,
191         pub_key: new_pub_key
192     }));
193 }
```

```
194
195   changeIssuerName(string issuer_id, string new_name, bytes signature) {
196       assert(owner == msg.sender);
197       assert(issuers[issuer_id]);
198       pub_keys = issuers[issuer_id].pub_keys;
199       assert(verifySignature(signature, new_name, pub_keys[pub_keys.length
      - 1].pub_key));
200       issuers[issuer_id].name = new_name;
201   }
202
203   setMinWorkerBalance(uint balance) {
204       assert(owner == msg.sender);
205       min_worker_balance = balance;
206   }
207
208 }
209
```

The above pseudo-code assumes msg.sender as the one who is submitting the transaction to the blockchain, and msg.owner as the owner of the smart contract.

4.1. Worker accounts

Nonce Management: A transaction nonce[14] is a sequence number associated with the transaction from an address that tells the blockchain in which sequence to execute the transactions. If an address is submitting multiple transactions, it must ensure to submit nonce with proper sequencing. With such strict nonce requirement for each transaction, it also makes it hard to monitor the submission of multiple transactions to the blockchain.

- A nonce gap is formed when one of the transactions doesn't make it to the network in the multiple transactions submitted.
- Many transactions can starve if one of the transactions is aborted because such an aborted transaction's nonce either needs to be canceled or re-used.

With the uncertainties that already come with gas price and mining, this nonce management is a huge pain for throughput since it limits the number of transactions an address can submit to the blockchain at once.
To overcome this, the LegitDoc smart contract uses worker accounts. Worker accounts can be dynamically increased by using the *addWorker* transaction. These worker accounts in turn are responsible for any other modifications done on the smart contract for issuance. Different worker accounts mean different nonce sequences. If we use a simple sequential nonce manager which waits for the current transaction to be acknowledged before submitting a new one, each worker account can be associated with a corresponding sequential nonce manager, hence parallelizing the issuance process. This coupled with batching at the issuer end as well as at the LegitDoc engine end should generate high throughput with just a few workers. This system is more efficient than when

an individual issuer interacts directly with the blockchain to issue documents because when the issuer is doing such a direct transaction, his time to finality and throughput is inherently restricted by the number of nonces he can submit and manage at once.

$$throughput \propto (N_f \cdot N_b) \cdot N_{workers} \tag{2}$$

where N_f is the total number of documents that can be packed in a single batch at the issuer end, N_b is the total number of documents batched at the server end, and $N_{workers}$ is the number of workers accounts available.
N_b can be significantly increased by scaling the server resources, and hence producing higher throughput.

Security: There has been lot of instances where a third-party blockchain service provider has been hacked and their keys have been compromised, leading to extreme situations[15] and hence compromising end-user security. Such instances happen because the digital signature private keys of third-parties need to be stored on the cloud to provide the service, and the cloud stack may itself be vulnerable to attacks that leads to leaking the private key. Fortunately, LegitDoc has a multi-sig mechanism where the issuer produce their own digital signature to make a document authentic. To further reduce the attack surface, LegitDoc uses the concept of worker accounts and owner account.
- Owner account is simply the owner of the smart contract that uses a multi-sig hardware wallet.
- Worker accounts are dynamically added accounts to the smart contract that can execute limited type of transactions.
With such a setup, the private keys of worker accounts can be used on the cloud to execute limited type of transactions like *invalidateSingle* and *putRoot*.
The owner account is an offline multi-sig hardware wallet account and guards any other entry-point on the smart contract. For example, owner account's manual multi-sig is needed to roll an issuer's key or invalidate an entire time window.

4.2. Tank Refilling

With the introduction of worker accounts, managing currency becomes harder since all the worker accounts need to have sufficient balance to execute transactions. To maintain this more easily, the smart contract can be made the single source of balance management. Whenever a new worker is added, the smart contract ensures the worker has a minimum balance, and whenever the worker transacts with the smart contract, the smart contract refunds the fee spent on the transaction to the worker account address. This way, it's enough to just refill the smart contract address with enough balance and the smart contract ensures the workers have enough balance to carry out the transactions.

4.3. Rolling Keys

Since issuers are organizations, and organizations are managed by people, and such people might be transferred from the position of authority, it's important to provide the ability for the issuer to change their digital signature key-pair. The smart contract supports rolling the keys by accepting a new public key and a signature that can be verified by the latest known public key of the issuer. The Issuer interface provides the user interface for the issuer to roll their keys.
Similarly, the issuer can also change their name by submitting a new name and signature that can be verified by the latest known public key of the issuer.

4.4. Social Recovery of Issuer Private Key

In cases of theft or loss of issuer's private key, the issuer can recover their private key using social recovery [16] that is supported on the smart contract. During issuer registration, the issuer can add three or more multisig public keys to the smart contract which are responsible later for voting and assigning a new public key on the chain to the issuer. In real-world terms, such multi-sig public keys can be given out to different people of power in the issuer's organization to prevent abuse of authority.

5 Conclusion

In comparison to centralized solutions, Decentralized solutions are inherently better positioned to solve the challenge of tamperproof credential/transaction verification. Such decentralized solutions however, due to their technical prerequisites today - are not positioned for uncomplicated customer adoption, seamless plug and play integrations with existing applications and provide the scalability requirements that an enterprise demands.

LegitDoc aspires to become the go-to platform for decentralized verification of digital documents and transactions, a platform which provides flexibility for customer adoption and seamless application integration; most importantly, a platform which provides tamper proof document authentication and real time verification at scale with globally decentralized public blockchains as the underlying foundation. LegitDoc platform's major components – customer centric Issuer and verification Interface, scalable LegitDoc Engine, open sourced LegitDoc Smart Contract; are carefully designed and developed to embrace decentralized application adoption by enterprises. The LegitDoc platform currently has two adoption ready service delivery models in place i.e., customer issuer and verification interfaces via a desktop and web application, with a third API first SDK service delivery under development.

Today, LegitDoc platform runs on production environments for public and private enterprises in industries such as government, education and IT (examples include

Maharashtra state board of skill development). Given the universal use of digital documents/credentials and the need tamperproof verification, the LegitDoc platform is industry agnostic and aims for customer adoption in banking and financial services, manufacturing, retail and healthcare.

Finally, to accommodate adoption from aspirational sectors, and to accommodate customer feature requests, the LegitDoc team is researching into the following as part of future considerations for the LegitDoc platform:

- **Industry-specific document designers** – Providing credential issuers an ability to design/format documents based on industry specific norms as part of the LegitDoc platform.
- **Automated document delivery** – Providing credential/document issuers an ability to deliver tamperproof blockchain anchored LegitDoc documents to credential users/stakeholder via email automation.
- **Document Storage** – Providing issuers an ability to outsource document storage and management via the LegitDoc platform as a low-cost alternative to customer self-storage.
- **LegitDoc Block Explorer** – Providing a public facing interface to appropriate LegitDoc stakeholders, giving them the ability to actively monitor their LegitDoc application usage and transaction insights.

6 Future Considerations

6.1. Soul-Bound Tokens (SBTs)

Soul-Bound Tokens (SBTs) proposed in May 2022 [17] lays out the foundation for a Decentralized Society. From this centralized society to a Decentralized Society is a long journey from web2 to web3. LegitDoc thrives to increase user adoption & awareness, and hence onboard people and organizations on that long journey that's headed towards issuing all documents as SBTs on a soul's storage. This makes LegitDoc one of the pioneering feasible use-cases for SBTs.

6.2. Governance Tokens

In the current model, the registration of Issuers' priavte key on LegitDoc smart-contract is done via a multi-sig smartcontract owner-private-key by LegitDoc developer team. In addition to the present decentralization measures taken, the issuer registration module as well can be fully decentralized by introducing a governance token that decentralizes the ownership and responsibilities of smart-contract owner-private-key .

References

[1] HuffPost. Uidai's aadhaar software hacked, id database compromised, experts confirm, 2018. URL https://www.huffpost.com/archive/in/entry/uidai-s-aadhaar-software-hacked-id-database-compromised-experts-confirm_a_23522472.

[2] Slate.com. How a 2011 hack you've never heard of changed the internet's infrastructure, 2016. URL https://slate.com/technology/2016/12/how-the-2011-hack-of-diginotar-changed-the-internets-infrastructure.html.

[3] thehackernews.com. Mongolian certificate authority hacked to distribute backdoored ca software, 2021. URL https://thehackernews.com/2021/07/mongolian-certificate-authority-hacked.html.

[4] Zdnet.com. New 'shadow attack' can replace content in digitally signed pdf files, 2020. URL https://www.zdnet.com/article/new-shadow-attack-can-replace-content-in-digitally-signed-pdf-files/.

[5] Simon Rohlmann, Vladislav Mladenov, Christian Mainka, and Jörg Schwenk. Breaking the specification: Pdf certification. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1485–1501. IEEE, 2021.

[6] Balaji Srinivasan. Creating sources of definitive truth with blockchain oracles, 2021. URL https://www.youtube.com/watch?v=Cwbbxb987vE.

[7] Joshua Ledbetter. What problem does web3 solve, anyway?, 2022. URL https://mirror.xyz/0xE4f646F0Be4fF5ce185540F5366295f91d75b65D/-xpmr7ceHmi5Hqsl7zRtig9ph_dtCvWjZOoWOVN0bcg.

[8] Neil Martis. Winning the war on document forgery in the digital economy, 2021. URL https://blog.legitdoc.com/winning-the-war-on-document-forgery-in-the-digital-economy-9fe3c86d5112.

[9] Fortune Media IP Ltd. Once billed as a rising star in crypto, solana's sixth outage this month, 2022. URL https://fortune.com/2022/01/25/solana-founder-anatoly-yakovenko-crypto-crash-blockchain-instability/.

[10] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger eip-150 revision.

[11] Georg Becker. Merkle signature schemes, merkle trees and their cryptanalysis. *Ruhr-University Bochum, Tech. Rep*, 12:19, 2008.

[12] Morris J Dworkin et al. Sha-3 standard: Permutation-based hash and extendable-output functions. 2015.

[13] S Blake-Wilson and M Qu. Standards for efficient cryptography (sec) 2: Recommended elliptic curve domain parameters. *Certicom Research, Oct*, 1999.

[14] The Startup. Ethereum series — understanding nonce, 2019. URL https://medium.com/swlh/ethereum-series-understanding-nonce-3858194b39bf.

[15] CoinGeek.com. Nicehash returns 100% of stolen funds, 2020. URL https://coingeek.com/nicehash-returns-100-of-stolen-funds/.

[16] Vitalik Buterin. Why we need wide adoption of social recovery wallets, 2021. URL https://vitalik.ca/general/2021/01/11/recovery.html.

[17] E Glen Weyl, Puja Ohlhaver, and Vitalik Buterin. Decentralized society: Finding web3's soul. *Available at SSRN 4105763*, 2022.