The Full Transparency System

verizon

verizon
Setting the Standard for Transparency
There's an overabundance of information, but not nearly enough tools to help determine what's trustworthy. By incorporating blockchain technology, Full Transparency Initiative reinforces our

Overview

As one of today's leading tech companies with a core goal of connecting people to each other and information, Verizon understands that innovation is continuous and success depends on the ability to build trust with the public. The pursuit of a better newsroom experience led us to explore ways to reinforce our commitment to transparency.

Full Transparency incorporates blockchain technology in an effort to set the standard in corporate communications. Blockchain enables us to securely and unalterably publish news releases. The technology helps create a definitive log of published content and any changes made to that content. The functionality and design used in the technology enable clear distinctions between tracked and untracked text content. Visual queues highlight the changes made to published news releases, including minor linguistic edits and statistical updates, as well as major factual corrections.

Purpose

This paper outlines:

- How Full Transparency's protocols are stacked
- What technology stands behind this blockchain system
- What public key infrastructure (PKI) is used within the system
- How many primary certificates are used in the PKI
- How timestamp proofs are validated
- Which hashing function is used in the system
- Which specific accumulator is used in the system
- How system users can commit articles to the blockchain
- Best practices for displaying committed articles and history to the end-user.
- How to get in touch to learn more about and utilize this technology in other organizations

Stack

Full Transparency exists as a collection of layered protocols, and each of these layers is oblivious to the layer above. This design choice was made to allow the system to adapt to future needs while only forcing a layer to perform functions deemed absolutely necessary.

The most fundamental layer of the system is a fully transparent record system. Within this context, that record system is a blockchain. The purpose of this ledger is to allow records to be written in an auditable system where the ordering of those records may be authenticated.

On top of this ledger is a system that allows the cryptographic keys of an organization to be publicly proven. This allows organizations to leverage those keys for the creation of auditable, timestamped digital signatures. This system is the public key infrastructure of **Full Transparency**.

Full Transparency is a non-repudiation system that leverages the public key infrastructure described above to create a repository of publicly auditable statements. These statements may be versioned and updated to accommodate the needs of enterprise operation.

The **Full Transparency** system leverages the timestamping capabilities of the underlying public key infrastructure to allow web browsers and mobile applications to authenticate the integrity of the data and origination of a piece of posted information without requiring direct access to the blockchain. Direct blockchain access is required to perform complete historical audits.

This layer of the stack is the application-specific tooling that may be run in the environment of an end user's device to verify the origination and recency of any publicly signed documents within a **Full Transparency** system. All of the formal treatment with respect to these systems may be found in the GutHub organization located here.

Open Source

The **Full Transparency** system is based on open-source tooling, and has been designed as a general purpose protocol that may work in many different settings. The Blockchain that stands behind **Full Transparency** is the MadNetwork blockchain. The technology that stands behind this blockchain is similar in nature to those technologies found in Certificate Transparency systems.

On top of this blockchain a public key infrastructure ("PKI") has been built by AdLedger. This PKI is based on a public standard and with an open source implementation. This general purpose PKI is the basis of non-repudiation within the **Full Transparency** system.

Open source tooling allows others seeking to adopt the standard being defined within **Full Transparency** to do so easily.

The Blockchain

Within **Full Transparency**, the blockchain acts as a verifiable datastore into which information may be written for auditability. The information that is written to the blockchain is the minimum amount of information that allows an auditor to know what information is being claimed by an entity, when that information was publicly claimed by an entity, the identity of the entity making these claims, and if any of the information claimed has been tampered with. This is the basis for **Full Transparency**.

Before the technical details of the **Full Transparency** system itself may be addressed, the infrastructure on which it operates must be introduced. This requirement is manifest from the fact that the blockchain provides the underlying data integrity protections for information written into the system but is ignorant of the **Full Transparency** system itself.

In order for information from the blockchain to be used in an auditable manner, users must know where to find all of the information written by each entity. Without some mechanism to find all information associated with an entity, audits of that data could not occur. The user must also know when each change to that information occurred, and that the information is protected from unauthorized modification. Fortunately, these requirements are directly aligned with the reasons blockchain technology was invented.

In order to address the issue of finding all information written into the blockchain by a specific entity, the concepts of accounts and account namespaces are used. An account is defined by the hash of a public key from an asymmetric key pair and the asymmetric algorithm of that key pair.

Each account owns an account namespace within the blockchain. An account namespace is a constructed index of all data written to the blockchain by a specific account such that that information may be easily accessed and tracked. Data may only be written into an account namespace by proving knowledge of the account private key using digital signatures.

Anyone may read the contents of an account namespace, but the rules of the blockchain have been constructed such that only the owner of an account may write data such that it will be indexed as a member of the owned account namespace. This rule is enforced through the use of digital signatures and the association of asymmetric keys pairs with accounts.

Every account namespace acts as an isolated, iterable, key-value store with the ability to hold up to 2256 keys. Due to this construction, any piece of data written into the blockchain may be uniquely identified by the account that owns the namespace in which the data was written and the key at which the data exists.

Each time information is written into an account namespace, the owner of the namespace must construct a signed transaction of the changes and data to be written. These signed transactions must be sent to the blockchain for inclusion in a block. Once included in a block, users may establish when a change in an account namespace occurred using the immutable history of the blockchain. The user may also be confident that only the owner could have written the change due to the use of digital signatures. Unfortunately a digital signature only proves knowledge of the private key from an asymmetric key pair. As such, another means to link the real world entity to the public key of the key pair is necessary.

Identity Verification

To address the issue of identity, an application specific PKI has been developed. PKIs provide a solution to the problem of linking a real world entity with a digital identity. This is accomplished by allowing a set of trusted actors to be the authority of digital identity within some domain.

Granting the authority to assign digital identity to a small set of actors makes proving a digital identity much more simplistic, but that same authority may also be abused to wrongfully link digital identities to real world entities without the knowledge or consent of the entity. To combat this abuse vector, the blockchain itself has been leveraged to provide auditability and accountability for issuance activities of the authorities as well. The complete details of this system are beyond the scope of the current document. The PKI that has been constructed operates on a short-lived certificate model. The following section addresses additional information about the certificates.

Certifications

Certificates in the **Full Transparency** system allow an end user to validate that a public key and/or blockchain account is an acceptable form of digital identity for a real world entity. The validation process of these certificates is analogous in nature to the process of verifying a SSL certificate chain, but the certificates themselves are based on the JOSE standard that underpins OAUTH technologies.

A validated certificate is a proof of ownership for the blockchain account namespaces themselves due to the inclusion of both accounts and public keys in these objects.

In this way, statements written to the blockchain may be trusted as having a provable origination. The certificates also name blockchain account namespaces where the parent certificates must be written to the blockchain to be considered valid. This requirement provides a unique solution to both the problem of Certificate Transparency as well as Certificate Revocation. Any certificate not written to the chain must be treated as invalid. This also implies all children certificates are also invalid. Thus, AdLedger is incapable of issuing a fraudulent certificate that is not an auditable element of the blockchain. Further, since all certificates must exist in the blockchain to be treated as valid, revocation may be performed through modification of the blockchain.

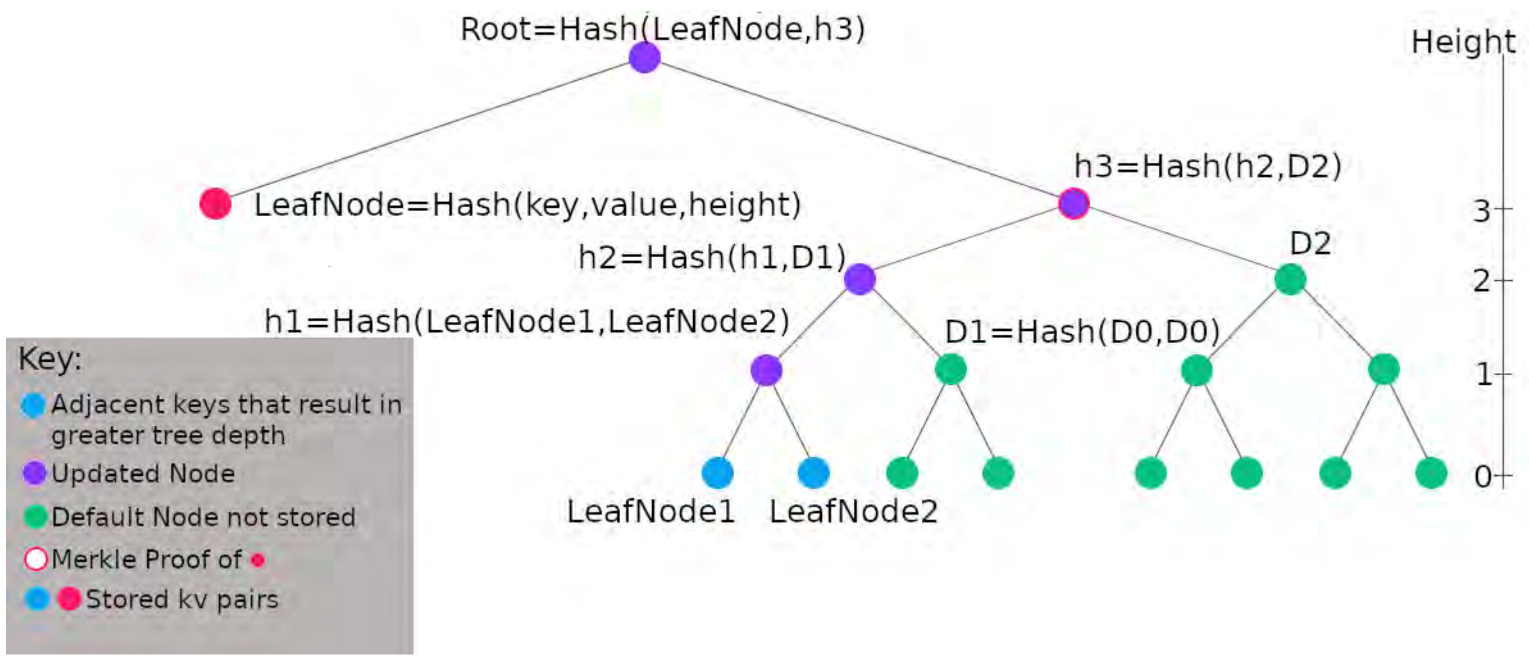
The discussion up to this point has focused on how the blockchain is constructed, how this construction facilitates the construction of owned namespaces and how identity is established for those namespaces. The data that is written to the blockchain will be addressed in the following sections.

Document Hashing

When a document is intended to be published with **Full Transparency**, a cryptographically secure fingerprint of the data must be formed. This fingerprint must be constructed using a cryptographically secure hashing function. The hash function used by **Full Transparency**, at this time, is Sha256.

The fingerprint may be built in one of two ways. The first way it may be constructed is by hashing the entire contents of the data using a cryptographically secure hashing algorithm.

Document Hashing



The second way a fingerprint may be formed is by construction of a Merkle Tree. Specifically, the tree implementation used is a Compact Sparse Merkle Tree based on the open source Aergo State Trie. We selected this implementation based on the ability to insert values at specific locations, the highly optimized logic, the ability to provide both proofs of inclusion, the ability to provide proofs of exclusion, and the concurrent nature of the underlying algorithm. In order to insert the data into a Merkle Tree, the data must first be split into chunks. At this time, the chunking algorithm is based on whitespace. The original content is fed through the chunking algorithm to generate an array of chunks. Each of these chunks is then hashed using Sha256 in order to generate an array of hashes.

Thus, for each chunk, there is a single hash. These hashes are then inserted into the Merkle Tree such that the keys of the tree represent the index in the array of chunks and the value is the hash of the chunk. The resultant root hash of the tree is treated as the fingerprint of the associated data, and this fingerprint is the only hash that is stored. Although this operation is far more complex than the first operation of simply hashing the data, the result is that subsets of the data may be proven using Merkle Proofs. The motivation for this capability is discussed in the future work section of this paper.

Non-Repudiation Objects

For each set of information that is to be tracked in the blockchain, a non-repudiation object must be constructed. These objects are stored along with the original content for all versions. These objects, in addition to a proof object concerning the state of the blockchain may be used to prove the information claims are valid on resource-constrained systems. These objects are based on the JWS standard. This document does not focus on the header of these JWS objects. The header section is concerned with validation of the PKI components of the system. The claims of the JWS objects are those elements directly related to the core focus of this document.

The Claims object contains the following fields:

1. JTI
2. Revision
3. IssuedAt
4. Exp
5. MMD
6. Chunking
7. MerkleRoot
8. Classification
9. PreviousRevNum
10. HeadURI
11. NodeURI
12. ContentURI
13. PreviousContentURI

The JTI field is set to the fingerprint of the version zero information. This serves as an identifier for all future versions as well. The Revision field is a monotonically increasing integer that is incremented for each new version of the document. The MerkleRoot field stores the fingerprint of the current version. IssuedAt is the unix timestamp for the time of creation of the object. MMD is the Maximum Merge Delay. Exp is the unix timestamp after which the object must be treated as invalid.

The MMD allows a delay between when an object is constructed and when it is tracked in the block chain. During the time from IssuedAt until the time defined by IssuedAt + MMD, a non-repudiation object may be treated as valid even if it does not exist in the chain. The HeadURI specifies a URI at which the latest revision exists for the given JTI. The HeadURI specifies a URI at which the minimum revision number may be requested for the given JTI. The minimum may not always be zero due to the requirement that some versions may be omitted from the record, but proof of their existence may never be removed. The NodeURI is a URI that may be called to return the maximum revision number may be requested for the given JTI. The ContentURI specifies a URI at which the raw content of the version of the associated Claims object may be requested. The PreviousContentURI and PreviousRevNum follow the same conventions as the other URI objects. It is of note the URI objects are not actively used at this time but have been included for future reference in other settings. The Classification field is, at this time, always set to PUBLIC. This field may be used to track if the non-repudiation object should be public domain. The Chunking field specifies the chunking algorithm used when forming the hash. algorithm.

Blockchain DataStructures & Versioning

In order to accommodate the fact that a document may be revised after initial publication, each document is tied to a unique identifier. This identifier is the fingerprint of the content from the first version of a **Full Transparency** statement. This identifier is held constant across all versions of the content.

As previously stated, the reason the revision number is not a strictly monotonically increasing integer is to accommodate the circumstance that a revision was made in error and must be stricken from the record. This action may be made apparent in the record through a jump in revision number greater than one. In this way, content that must be removed for legal reasons may be removed gracefully without breaking the protocol while also allowing transparency with respect to the fact that a revision has been omitted from the record.

The blockchain itself stores the MerkleRoot at the location identified by the JTI and Revision. This object also stores the next, current, and previous valid Revision numbers for iteration of revisions. In addition to these objects, a head object is also written to the blockchain namespace. The head object is written to the location of the JTI for the document. The head object identifies the latest revision number of a document at any point in time. Thus, in order to iterate the full revision history of a document, the head object may be parsed for the latest revision number. Given the latest revision number and a document identifier each object may be read from the blockchain namespace one node at a time. Since each node references the previous valid revision number, each prior node may be traversed until a node is found that has a previous revision number and current revision number that are equal. This node represents the tail of the linked list that defines the revision history of a document. The associated JWS objects may be requested from the issuer of the objects along with the content for audit purposes.

Future Work

The future work for this project is to enable those features that have been designed to allow third party validation of information in a trustless manner. In order to accomplish this goal, several features have been included.

First, in order to allow devices to validate data from the blockchain a novel certificate proof system has been constructed. This proof system also allows for the construction of short-lived proofs against the non-repudiation objects themselves. The combination of such proofs allows an verifying party to know that a certificate chain was valid at a specific point in time and that a non-repudiation object also existed during this same time frame.

The manner in which these proofs are constructed is by treating the blockchain namespace of all certificates and non-repudiation statements as a directed acyclic graph. This is possible due to the requirement that certificates do not perform cross-signing. Thus, a Merkle Tree may be constructed across all valid certificates and non-repudiation objects at fixed intervals in time. The root hash of this Merkle Tree is signed by a set of designated keys that are claimed in the intermediate certificates of the certificate chain. This signed object is then published into the blockchain such that an observing party may construct proofs of inclusion against the Merkle Root.

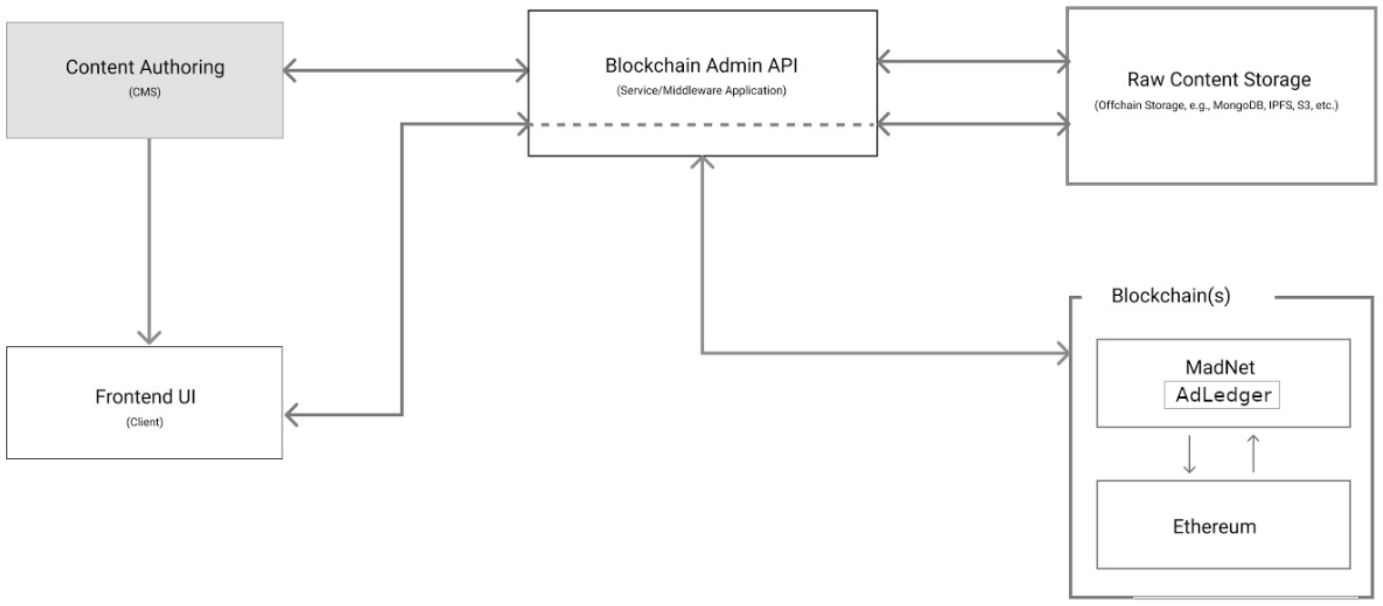
These proofs of inclusion may be constructed by any party in a trustless manner since the root hash itself is signed by the certificate authority. This is analogous to an Online Certificate Status Protocol with transparent stapling. The eventuality of combining these proof objects with the Merkle Tree based document hashing, is to allow a third party to prove a sub quote of a claimed document in the context of a third party system.

Presentation and Implementation

Full Transparency is engineered in a way that is intended to work with any number of content authoring/managing systems and any front-end web or app technology. By building API layers that an existing website can communicate with, the authoring of content, storage of content, and blockchain tasks can work in parallel without disrupting the current architecture of existing apps or websites–and run on the same or different infrastructure if needed. This section describes the general architecture and functions of the authoring, content storage, and presentation (Frontend UI) aspects that make up **Full Transparency.**

Architecture

Example:



General example of a decoupled architecture.

As shown in the basic architecture diagram above, the **Full Transparency** system has been designed to separate concerns and decouple the authoring, presentation, content storage, and blockchain systems.

It is recommended that any system that incorporates **Full Transparency** follow the same base principles.

Authoring

The authoring system should be aware of and commit data when:

- Adding new content
- Revising existing content

It is highly recommended that any content that is considered "published and publicly available" be committed to the blockchain as new content or as a revision to existing content. Depending on the content management system used, this can be achieved programmatically by calling a decoupled "blockchain admin API" at publishing time via a hook or direct function call for example.

When handling situations where some content may be considered "on-chain" and committed and other content will not be committed in the same content management system, it is important to make that decision at the text content's inception. This ensures that the complete history of that content is tracked. It defeats the purpose of transparent communications to retroactively add content to the blockchain without notifying users either that:

1. The content pre-dates the installation of the **Full Transparency** system
2. For some reason, the article was not committed at inception and there is an immediately accessible explanation available to the frontend user.

Content Storage

A key infrastructure element is the ability to maintain a record of every version of each article, in its entirety. When articles are written or edited and then committed to the blockchain, the raw data goes through a hashing process and the original article content must be stored (un-hashed) off-chain, in an independently maintained storage location where it will persist and be accessible (readable) to all systems.

The offchain storage system may be built on any number of technologies to store the raw text or content data, e.g, MongoDB, AWS S3, Google Cloud Storage, Interplanetary File System (IPFS). As long as the system(s) used for storing data offer a valid URI that is publicly readable, the system can store and use a single or multiple set of URI's to link the hashed commit with the un-hashed raw data. Off-chain storage is a redundant storage system with the capability of holding multiple storage buckets (e.g., servers) at the same time.

It is advisable, while all content and its history (past versions) will be stored off-chain the current (latest) version of the content that is served to the frontend should continue to live within the content management ecosystem (or other content server) to maintain frontend performance when not accessing the full history.

The off-chain storage exists as a repository to ensure article data persists with redundancy, for the purpose of checking published versions with the last, blockchain committed version for accuracy, and for being able to read and compare prior committed versions of any given piece of content.

Frontend Presentation

The frontend (website or app interface) is responsible for the general presentation of the data. The blockchain data exists 'under the hood' and it is the responsibility of the frontend to be 'transparent' and represent the data in its entirety. Best practices here include:

- Show all revisions and their data. No content should be omitted.
- Display the timestamp.
- Ensure users can clearly see the difference between versions.

How the historical data is displayed and highlighted can be determined by the authoring party. Inline or side-by-side comparisons that make the changes easily understandable are highly recommended.

If you or your organization are interested in utilizing blockchain technology to help build credibility and transparency in publishing content, please get in touch at fulltransparency@verizon.com