Secure Asset Transfer Protocol
Internet-Draft
Intended status: Informational
Expires: 19 September 2026

V. Ramakrishna
V. Pandit
IBM Research
E. Abebe
Consensys
S. Nishad
K. Narayanam
IBM Research

18 March 2026

Views and View Addresses for Secure Asset Transfer
draft-ramakrishna-satp-views-addresses-07

Abstract

With increasing use of DLT (distributed ledger technology) systems, including blockchain systems and networks, for virtual assets, there is a need for asset-related data and metadata to traverse system boundaries and link their respective business workflows. Core requirements for such interoperation between systems are the abilities of these systems to project views of their assets to external parties, either individual agents or other systems, and the abilities of those external parties to locate and address the views they are interested in. A view denotes the complete or partial state of a virtual asset, or the output of a function computed over the states of one or more assets, or locks or pledges made over assets for internal or external parties. Systems projecting these views must be able to guard them using custom access control policies, and external parties consuming them must be able to verify them independently for authenticity, finality, and freshness. The end-to-end protocol that allows an external party to request a view by an address and a DLT system to return a view in response must be DLT-neutral and mask the interior particularities and complexities of the DLT systems. The view generation and verification modules at the endpoints must obey the native consensus logic of their respective systems.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://ietf-satp.github.io/draft-ramakrishna-satp-views-addresses/draft-ramakrishna-satp-views-addresses.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ramakrishna-satp-views-addresses/.

Discussion of this document takes place on the Secure Asset Transfer Protocol Working Group mailing list (mailto:sat@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/sat/. Subscribe at https://www.ietf.org/mailman/listinfo/sat/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-satp/draft-ramakrishna-satp-views-addresses.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Blockchain systems, especially those of the permissioned variety, are a heterogeneous mix, fulfilling a diverse range of needs and built on several different DLT stacks that are not compatible with each other. Yet, in an interconnected world, the business processes running on each system cannot afford to remain isolated and the virtual assets they manage cannot afford to remain in siloes. These systems must be interoperable so that assets can move, and their properties can be reflected across network boundaries, and so that business processes can span multiple systems. Interoperability will, in effect, mimic a larger or scaled up, blockchain system composed of smaller blockchain systems without explicitly requiring those systems to coalesce.

At the core of any cross-blockchain system transaction is the ability of a system to project a view of assets and associated data recorded on its ledger to external parties, be they individual agents or other blockchain systems. On the reverse, a blockchain system must also have the ability to identify and address views of assets or associated data in another system, and further, to validate that view before its business process consumes it. View projection, addressing, and consumption, must eliminate or minimize the role of third parties to avoid loss of data privacy, control over business process, or diminishment of autonomy.

This document specifies formats for views and view addresses on distributed ledgers. Similar to the notion of database views, distributed ledger views reflect what a given network wishes to expose to external parties, typically through scripts, as well as access control considerations. In contrast to conventional databases, exposure and access control considerations must be decided through decentralized consensus. Also, in contrast to a conventional database, the consumer of the view, who may be a DLT system, must be

able to independently validate it as the consensus view of the providing network. Hence, a view incorporates the notion of a proof made against a claim. The view address must encapsulate the view request, and optionally carry a policy that circumscribes the construction of proof.

The protocol to communicate view requests and responses is beyond the scope of this draft and will be specified in a separate draft.

2. Terminology

The following are some terminology used in the current document:

* Blockchain system: Blockchains are distributed digital ledgers of cryptographically signed transactions that are grouped into blocks. Each block is cryptographically linked to the previous one (making it tamper evident) after validation and undergoing a consensus decision. As new blocks are added, older blocks become more difficult to modify (creating tamper resistance). New blocks are replicated across copies of the ledger within the network, and any conflicts are resolved automatically using established rules [NIST].

* Blockchain network: This is a blockchain system that is built on a network of nodes that maintain a common shared copy of the blockchain using a consensus protocol.

* Distributed ledger technology (DLT) system: Technology that enables the operation and use of distributed ledgers, where the ledger is shared (replicated) across a set of DLT nodes and synchronized between the DLT nodes using a consensus mechanism [ISO]. Every blockchain system is also a DLT system, and so we will mostly use the latter term in this draft when discussing protocols for cross-system interactions.

* Distributed ledger network: This is a DLT system that is built on a network of nodes that maintain a shared copy of the ledger or portions of it on different subsets of nodes using a consensus protocol.

* Resource Domain: The collection of resources and entities participating within a blockchain or DLT system. The domain denotes a boundary for permissible or authorized actions on resources.

* Interior Resources: The various interior protocols, data structures and cryptographic constructs that are a core part of a blockchain or DLT system. Examples of interior resources include

the ledger (blocks of confirmed transaction data), public keys on the ledger, consensus protocol, incentive mechanisms, transaction propagation networks, etc.

* Exterior Resources: The various resources that are outside a blockchain or DLT system, and are not part of the operations of the system. Examples include data located at third parties such as asset registries, ledgers of other blockchain/DLT systems, PKI infrastructures, etc.

* Nodes: The nodes of the blockchain or DLT system which form the peer-to-peer network, which collectively maintain the shared ledger in the system by following a consensus algorithm.

* Gateway nodes: The nodes of the blockchain or DLT system that are functionally capable of acting as a gateway in an asset transfer. A gateway node conforms to the Secure Asset Transfer Architecture [SATA] and implements the Secure Asset Transfer Protocol (SATP) [SATP]. Being a node on the blockchain/DLT system, a gateway has at least read access to the interior resources (e.g., ledger) of the blockchain. Optionally, it may have write access to the ledger and also the ability to participate in the consensus mechanism deployed for the system. Depending on the blockchain/ DLT system implementation, some or all of the nodes may be gateway-capable.

* Clients: Entities are permitted to invoke smart contracts to read or update ledger state in a blockchain or DLT system. They possess unique identities in the form of public keys. In a permissioned system, identity certificates are issued by the system’s internal providers (or certificate authorities).

* Wallets: Collection of client identities represented by public-private key pairs, and optionally certificate issued by a blockchain or DLT system’s identity providers (or certificate authorities).

* Virtual Asset: A virtual asset is a digital representation of value that can be digitally traded, or transferred as defined by the FATF [FATF].

* Virtual Asset Service Provider (VASP): Legal entity handling virtual assets as defined by the FATF [FATF].

* Ledger state: A snapshot of the information held in a distributed shared ledger, typically (though not necessarily) as a set of blockchain or DLT system. Examples of interior resources include key-and-value pairs. This information includes records of virtual

assets and the state of business processes that are meaningful to the DLT system's participants and clients. State elements and subsets may be scoped under the namespaces of given smart contracts, thereby being accessible only through invocations on those contracts.

* Smart contracts: Business workflows written in programming languages that manage the states of assets and business processes on a shared ledger in a DLT system. These contracts constrain the ability of system clients to modify ledger state and embed guards around state elements. Contracts can be invoked to read from, or write, to, a ledger. They can be deployed on several system nodes, who must agree on a given ledger state update via a consensus protocol.

* Consensus protocol/mechanism: Process by which nodes agree on a ledger state update, typically (though not mandatorily) through a smart contract invocation.

* Local transaction: A transaction triggered by a client to update the ledger state in a blockchain or DLT system, typically (though not mandatorily) through a smart contract invocation.

* View: A projection of a blockchain or DLT system's ledger state for external consumption, i.e., for parties outside that system. This can be a single element, a subset of the state, or a function over a subset.

* View Address: A unique identifier or locator for a view into a blockchain or DLT system's ledger. This is analogous to an HTTP URL.

* Source system: Blockchain or DLT system governing the ledger from which a view is produced.

* Destination system: System in which a view is consumed. This can be a blockchain or DLT system.

* Remote system: Counterparty system in a view request-response protocol instance. From the vantage point of the source system, this refers to the destination system, and vice versa.

* View requestor: Person or organization triggering a view request from a source network.

* Proof: A data structure containing evidence linking a view to its source system's blockchain, or more generally, ledger. The evidence may be probabilistic and in some cases cryptographically verifiable.

* Access/exposure policy: Set of rules governing the release of a view to an external party (i.e., outside the source system), held in consensus by nodes on the source system's ledger.

* Verification policy: Rules for validation of proofs associated with views maintained in a destination system. If the destination system is a blockchain or DLT system, these rules are held in consensus by nodes on that system's ledger.

* View Request: A request for a made by an external party to a source blockchain or DLT system. The external party may be a client in a destination blockchain or DLT system. The request consists of a view address and various metadata, including optionally a verification policy.

* Asset locking or escrow: The conditional mechanism used within a blockchain or DLT system to make an asset temporarily unavailable for use by its owner. The condition of the asset release can be based on a duration of time (e.g., hash time locks) or other parameters.

Further terminology definitions can be found in [NIST] and [ISO]. The term 'blockchain' and 'distributed ledger technology' (DLT) are used interchangeably in this document.

3. Assumptions and Principles

The addressing of a DLT system' s assets and asset-related data as well as the exposure of such data to a foreign DLT system assumes the presence of one or more gateways that are either part of the system or working on behalf of it. These gateways are ultimately responsible for generating and interpreting both addresses and data, hiding any DLT- or network-specific protocol considerations from each other. All DLT- and network-specific software artifacts that are exchanged among gateways will be encapsulated in generic (or DLT-neutral) software artifacts. The gateways are assumed to be interoperable using a protocol that corresponds to the design principles of the Internet architecture. The specification of this protocol is beyond the scope of this draft and will be described in a separate draft.

4. Ledger State Views and Proofs

4.1. Abstraction of Distributed Ledger States

A distributed ledger system maintains a set of ledgers, akin to databases. Each such ledger is organized and addressable in a hierarchical namespace with the identifier of the distributed ledger system being the root. A ledger is shared among a subset of members of the network that the system, which maintains the distributed ledger, is built on. These subsets of members may overlap or be mutually exclusive, depending on the precepts of the given DLT, but for the purpose of this document, the distinction is not relevant. For example, the Hyperledger Fabric blockchain platform maintains a set of channels, where each channel is shared exclusively by a subset of members [Andr18]. In contrast, the Corda platform allows each data item to be shared by an arbitrary subset of members, in effect creating databases privy to mutually overlapping member sets [Hear19]. The Ethereum Mainnet has a dynamic group of members maintaining a single global ledger with open, or public, access.

Each shared ledger within a system maintains a snapshot of the latest, or current, state, determined through consensus (or agreement) by the members sharing it. In addition, a tamper-evident history of the current state, which can be a blockchain or any other form of a transaction log, is also maintained by the members sharing that database. The state of the distributed ledger system as a whole is the union of states of the ledgers it comprises of.

Finally, any data item within a ledger can be retrieved, or any processed data extracted, using its native logic and interfaces. As a ledger is a traditional database in most respects, it supports extraction and processing the same way a database does. For example, data in a SQL database can be extracted using record keys and schema attributes, whereas in a No-SQL database, data is extracted using knowledge of key value pairs and overlaid schema. Just as views and stored procedures are used to extract information from a database, UTXO scripts (in Bitcoin [Naka08]) or smart contracts (in Ethereum [Ethe22], and several permissioned DLTs like Hyperledger Fabric and Corda) are used to extract information from a ledger. An interface is provided to give the user the ability to query or update ledger data. As UTXO scripts are just simple forms of smart contracts, we will assume in our abstraction that each ledger within a DLT system possesses a smart contract interface.

4.2. Distributed Ledger System Views

A view into a distributed ledger system state is similar in concept to a traditional database view but has certain additional features because the backing state is maintained in a different way than state in a traditional database is. Fundamentally, a DLT system view is information that is derived from that system's ledger. We define it as an abstract representation of state projected from a ledger that is consumable by entities, who may or may not belong to the network or possess credentials to access raw ledger state. Views are uniquely addressable within a DLT system. A view can be static, i.e., referring to information derived from a point-in-time (or historical) state, or dynamic, i.e., referring to information derived from the current state.

Semantically, a view represents the what and not the how, i.e., it projects information from a ledger but not the details of the process through which that information came into being. This process has multiple facets, from the consensus and commitment protocols through which raw data was recorded on the ledger to the smart contract procedure through which information was extracted from the raw ledger data. These procedures are specific to DLT implementations, which we wish to keep opaque from the cross-system communication infrastructure, specifically the systems' gateways. This will allow the communication infrastructure to ignore details of ledger implementations while managing state when a view is communicated from a system to an external entity. Therefore, we specify the view as a package with a generic structure, independent of a specific DLT implementation. Internally, a view will contain data that has a DLT-specific representation, but we will leave the interpretation of this to modules internal to the systems. In this document, we will specify the formats of the DLT-independent portions of a view that will be handled by the communication infrastructure and provide suggestive sample view contents for prominent DLTs.

There is a caveat to the above definition, arising from a fundamental difference between a traditional database and a DLT system. This is the fact that state in the former is maintained by a single authority whereas state in the latter is maintained collectively, and held in agreement, by a peer group of entities, each of which can fail or be malicious. Therefore, a DLT system view is incomplete without an associated proof of it being derived from state agreed to by the group as a whole. In practice, this does not require unanimity but rather enough representation from group members to overcome failure of individual peers' failures and to assure an external client that the system as a whole will not repudiate that view, in accordance with the consensus rules of the source ledger, at a later time. Theoretically, we can quantify the nature of this proof under certain

models. If peers are susceptible only to crash failures, an agreement over state from more than 50% of the peers will qualify as sufficient proof. If peers can be malicious, i.e., fail in Byzantine ways, a consistent state view is required from more than 2/3 of the peers. More generally, the nature of proof, or determining what is sufficient, can be derived from the consensus mechanism used by the system.

The proof associated with a view represents two things. One is an assurance that the view is a group, or consensus, view of the ledger held by the DLT system as a whole. The other is evidence of authenticity of the state projection that the view represents. And though the construction of a proof is very DLT-specific, the notion that a proof may be supplied with a view can be embodied in a DLT-independent specification, thereby adhering to the principle of DLT opacity to the communication infrastructure. Finally, proofs are also consistent with the "what, not how" principle because they certify that a view represents state at a given point in time and not the history of events that led to that state.

Now we can look at the structure of a view in more detail. At a high level, a view consists of the following:

* Request Id: a unique value corresponding to the request for a view made by an external entity to the DLT system.

* Data: ledger state projection, or derived information, with proof.

* Metadata: attributes of the payload used to unpack and interpret it.

Data consists of the following:

* View Address: this is supplied by an external entity, and is the equivalent of a "getter function" that can be used to uniquely identify (or derive) projection into a DLT system state.

* Information: this is the actual ledger state projection.

* Schema: this described the Information data structure and can be used to unpack (or unmarshal) it. It is optional if the ledger schema is well-known.

* Proof: This is a structure that validates the Information. It is optional, though recommended for DLT system views.

* Proof Schema: this describes the Proof data structure. It is optional if the proof schema is well-known.

Metadata consists of the following:

* View Type: this tells us whether the view is a singleton data item, a collection of data items, or a computation over data items that are part of the DLT system state.

* Protocol Type: this denotes a unique value associated with a given DLT protocol; e.g., Bitcoin, Ethereum, Hyperledger Fabric, Corda, Solana.

* Timestamp: this denotes the time at which the view was produced.

* Proof Type: this denotes the type, or category, of proof being supplied, e.g., Notarizations/certifications (also called proof of authority), Simplified Payment Verifications, Zero Knowledge Proof, Proof of Proof-of-Work.

* Serialization Format: this denotes the format used to serialize the view data, e.g., XML, JSON, protocol buffer.

* Commitments: these are commitments made on the view by authorities or infrastructure (e.g., the Ethereum Mainnet) external to the DLT system.

4.3. Simple Views

A simple DLT view is any unit of a database within a DLT system that is exposable through interfaces programmed over that database. More concretely, any blockchain or smart contract system provides the ability to write scripts or procedures that can act on the raw ledger (database) data, accompanied by interfaces to trigger those scripts or procedures to query or update ledger state. In such a system, a simple view is any information that can be obtained directly through an invocation of this interface, e.g., any transaction exposed by a smart contract deployed on a ledger within the system.

In the DLT view structure specified earlier, the View Type within Metadata will be set to SIMPLE if such a view is requested by an external entity. The content of the view address (described later in this draft) can be interpreted to know if the external entity is requesting a simple view.

4.4. Aggregate Views

An aggregate DLT view is a collection of addressable units of databases within a DLT system. In effect, it is an array of simple views, which can be derived through multiple invocations of different scripts or smart contract transactions acting on raw ledger data. In the DLT view structure specified earlier, the View Type within Metadata will be set to AGGREGATE if such a view is requested by an external entity. The content of the view address (described later in this draft) can be interpreted to know if the external entity is requesting an aggregate view.

4.5. Complex or Processed Views

A complex or processed DLT view is the output of a function computed over a set of addressable units of databases within a DLT system. In effect, it is a function computed over an aggregate view. In the DLT view structure specified earlier, the View Type within Metadata will be set to AGGREGATE if such a view is requested by an external entity. The content of the view address (described later in this draft) can be interpreted to know if the external entity is requesting a complex view, and the function to be computed will also be specified in the view address.

4.6. View Proofs

Proof within a view must ratify the view's contents as the consensus over a projection of ledger state by members of the DLT system. Because the projection of state is itself DLT-specific, though it can be wrapped in DLT-neutral structures as we have described earlier, the proof also takes DLT-specific shapes. But we can list the set of attributes any proof must contain in abstract terms as follows:

* Association of response with request: a connection (ideally, cryptographically secure) between the request supplied in the view address with the ledger state projection as inferred by the source system. For instance, this can take the form of a signature over a common structure consisting of both the request and the response.

* Authenticity of response: a connection (ideally, cryptographically secure) between a member generating a ledger state projection and its DLT system. For instance, this can take the form of a certificate chain associating the signer with the DLT system's membership authorities. This is necessary for a permissioned DLT system and is optional for open (or permissionless) DLT systems.

* Evidence of consensus: information showing that the view contents are agreed upon by the network as a whole. For a permissioned DLT system, this typically implies that an identical and authentic ledger state projection must be provided by a large enough quorum of its members so that the view cannot be repudiated in the future. In an open (or permissionless) system, this can simply be the portion of a mined block that shows why it belongs to the longest chain; i.e., proof-of work or succinct versions of it (like Proof-of-Proof-of-Work [Naka08], or PoPoW [Kiay16][Kiay17]). In any DLT system, such evidence can optionally pass a policy check, where the policy rule is supplied by the view requestor and typically embodies the consensus logic that led to the commitment of the ledger state projection being requested.

5. Ledger State View Addresses

To request a view from a DLT system, an external entity must be able to unambiguously specify the information it seeks in the form of a view address. The use of the term “address” follows from the abstraction of a DLT system as a repository of data resources that can be retrieved and computed on. A view address in blockchain or distributed ledger systems is equivalent to URLs [RFC1630] (for example, specified as an HTTP address [RFC2616]), which are used to address resources offered by Internet and World Wide Web servers [RFC1738].

From a functional perspective, a view address can also be thought of as an interface over a “getter”, which is a standard software pattern used to fetch data values in a computer program. The schematic representation of a getter is dependent on the protocol followed by the underlying distributed ledger system, but is conceptually abstracted away by the view address. An external entity can create and manipulate a view address without understanding its semantics and usage, which are hidden by the DLT system. This allows the system to use alternate representations for getters internally without requiring external entities to understand the implementations of these operators. The view address states the “what” and not the “how”.

A view address must be a globally unique identifier of a view on a distributed ledger system, because global interoperability is our goal. Therefore, as with DNS addresses for Internet servers and URLs for World Wide Web resources, a view address is a hierarchical address, with different segments identifying units of decreasing size and increasing specificity in sequence. The syntax is as follows:

```
address  = location-segment "/"
           ledger-segment "/"
           view-segment
           ; distributed-ledger-system/ledger/data-projection
```

The location-segment provides identification and reachability information for the distributed ledger system. The ledger-segment identifies a unique ledger within this system, and the view-segment identifies a view or state projection from that ledger.

Decentralized ledger systems don't have a single location as they are built on networks of peer nodes. Maintainers of these systems may expose one or more endpoints for external entities to access views, which can be hosted on the network nodes themselves or on designated gateways. Gateways form the communication infrastructure for cross-system interactions and can be deployed in different configurations: a single system may possess multiple gateways, or a single gateway may serve multiple systems. Therefore, to formulate a view address, one needs to know the set of endpoints that lead to a given distributed ledger system and a unique identifier for the network that hosts that system. In certain systems and gateways, an identifier that represents a set of endpoints can be used instead of explicitly enumerating those endpoints. Also, if the specified set of endpoints is known to serve a single system, the network identifier can be omitted. The syntax of the location-segment is as follows:

```
location-segment  = gateway ["/" network-id]
gateway           = endpoint *(";" endpoint) / name
endpoint          = host [":" port]
host              = hostname / IP-address
port              = 1*DIGIT
network-id        = name
hostname          = name 1*("." name)
name              = (ALPHA / "_") *(ALPHA / DIGIT / "_" / "-")
IP-address        = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
```

A single gateway serving a given distributed ledger system (network) can be represented as follows:

```
location-segment  = gateway.trade.com:7542/trade-network
                    \_____________________/ \___________/
                               |                  |
                           endpoint           network
```

Multiple gateways serving a network can be represented as follows:

```
location-segment  = gateway1.trade.com:7542;   \
                    gateway2.trade.com:7542;   |--gateway (endpoints)
                    gateway3.trade.com:7542    /
                    /trade-network
                     \___________/
                           |
                        network
```

Gateways serving a single well-known network can be represented as follows:

```
location-segment  = gateway1.trade.com:7542;gateway2.trade.com:7542
                     \_____________________________________________/
                                          |
                                  gateway (endpoints)
```

The ledger-segment names either the ledger, if that ledger has a distinct identifier within the system, or a procedure to access a ledger, which represents a common set of facts shared by a subset of members of the network that maintains the distributed ledger. These subsets may overlap; i.e., certain nodes may maintain more than one ledger in the system. The procedure name can take the form of an identifier that represents, say, a decentralized application spanning certain nodes, or an explicit enumeration of the identifiers of the nodes themselves. The ledger segment syntax is as follows:

```
ledger-segment = *1(
                    (ALPHA / "_")
                    1*(ALPHA / DIGIT / "_" / "-" / ";" /  “:” )
                   )
```

The ledger-segment can be blank if the distributed ledger system has a single ledger. This is the norm in open blockchain systems like Bitcoin or the Ethereum Mainnet, and in private Ethereum-based systems like Quorum and Hyperledger Besu.

Examples are as follows:

```
ledger-segment = trade-channel
                 \___________/
                       |
                  ledger name

ledger-segment = paymentsDapp
                 \___________/
                       |
               procedure/app name

ledger-segment = paymentsDappNode1:9005;paymentsDappNode2:9005
                 \_____________________/ \_____________________/
                           |                       |
                  procedure/app node     procedure/app node
```

The view-segment uniquely identifies a view within a ledger. Features particular to a distributed ledger technology will determine how to encode this segment, but we can draw out common abstractions across DLTs to create a generic specification. All such technologies offer a procedural interface to access and manipulate data, typically (but not always) in the form of a smart contract. The exposed interface offers multiple functions to generate views based on the provided input. Hence, we can specify the view-segment as being composed of a contract, a function, and a list of input arguments. In the most general case, a default contract may be assumed, and arguments may be unnecessary, and so these can be omitted. The function, which can either be a procedural identifier, or a direct reference to a data item or collection of data items, or a programming instruction, must be specified. The view-segment syntax is as follows:

```
view-segment   = [contract-id] ":"
                 function-spec *(":" input-argument)
contract-id    = (ALPHA / "_") 1*(ALPHA / DIGIT / "_")
function-spec  = name
input-argument = name
name           = *HEXDIGIT ; hex-encoded ASCII string
```

An example of a view-segment for a Hyperledger Fabric ledger is:

```
view-segment   = trade-chaincode:getBillOfLading:bill-10012
                 \____________/ \_____________/ \________/
                        |              |            |
                    contract       function     argument
```

An example for a Corda ledger is:

```
view-segment   = :com.trade.dapp.flows.GetDocumentByTypeAndId:C:5
                  \_____________________________________________/ \_/
                                          |                       |
                                       function               arguments
```

An example of a complete view address is as follows:

```
gw.trade.com:7542/traden/tradel/tradec:getBill:bill-10012
\_____________________/ \____/ \_____________________________/
           |               |                |
    location-segment  ledger-segment   view-segment
```

6. Ledger State View Verification Policies

A verification policy for a ledger state view is a set of rules that the proof within the view can be validated against (or filtered through). The condition embedded within a rule can be arbitrary, though in practice, it should embody the process by which that ledger's state was updated and consented to in a decentralized manner by the network of peers maintaining it. In permissioned networks built on DLTs like Hyperledger Fabric or Corda, a policy typically requires evidence of attestations made by a sufficiently large, or representative, portion of the peer network maintaining the ledger. This takes the form of a set of signatures and is crucial to validating views generated by networks built on DLTs like Hyperledger Fabric and Corda. In open networks, we can envision policies that require validation of the view data being derived from a block in the longest chain, for example. But in this specification, we will consider only attestation-based policies and leave more general policies to future drafts.

The structure of a verification policy is as follows:

* Security Domain: a unique identifier for the distributed ledger system (or network maintaining it) projecting the ledger state view.

* Rules: a set of rules, each governing the validation of artifacts exposed through a particular category of views within the Security Domain.

Each Rule consists of the following:

* View Pattern: a regular expression that can match a set of views.

* Policy: a policy rule/filter governing all views that match View Pattern.

Each Policy consists of the following:

* Type: an identifier or flag denoting the type of this policy rule. For attestation-based policies, this should be set to “Signature”, and other identifiers can be created for other policy types in future drafts.

* Criteria: a Boolean expression with ANDs and ORs, where the principals consist of (1) the name of a DLT system unit/ stakeholder (typically corresponding to a subset of nodes in the peer network), and (2) the number of signatures requires from this unit.

7. Ledger State View Request

A view request is a message sent to a distributed ledger system by any external entity. It consists of the following:

* View Address: uniquely identifies the data sought by the requester.

* Verification Policy: indicates the proof criteria for the requested view.

8. Ledger State View Access Control Policies

An access control policy for a ledger state view is a set of rules governing the exposure of the view to an external entity. Each rule maps a set of principals to a set of views. The structure of an access control policy is as follows:

* Security Domain: a unique identifier for the entity (which can be a distributed ledger system itself) requesting a ledger state view.

* Rules: a set of rules, each governing access from some entity within Security Domain to artifacts exposed through a particular category of views.

Each Rule consists of the following:

* Principal: a string that can match a set of subjects or principals, which can represents an individuals or an organization or a subgroup within an organization identified by role or attribute set (profile).

* Principal Type: a keyword that denotes the nature of the principal. This can be one of the following:

- PUBLIC-KEY: indicates that the principal is a public key associated with an individual member of the Security Domain.

- CA: indicates that the principal is a certification authority for an organization within the Security Domain.

- ROLE: indicates that the principal identifies a role within the Security Domain.

- ATTRIBUTE: indicates that the principal identifies a set of attributes, or a profile for a member, within the Security Domain.

- '*' : indicates that the principal can be any member of the Security Domain. The Principal can be left blank in this case.

* Resource: a regular expression that can match a set of views, which identifies the objects governed by this rule.

* Read: a Boolean flag indicating whether this rule is currently active.

9. Related Open Issues

This draft provides a specification for views and how to addresses them. It further describes a protocol whereby one system can request a view from another through gateways. But there are several aspects of the end to-end process, which are extraneous to the data sharing protocol yet crucial to its successful completion. Though detailed specifications of these are beyond the scope of this draft, we list them in this section for readers' considerations.

9.1. Forms of proof

Though we have tried to be agnostic of the nature of the proof associated with a view in this draft, the data sharing protocol implicitly assumes that the proof takes the form of a quorum of digital signatures from parties belonging to a permissioned distributed ledger system. But several other proof types can exist, each suitable to the type of system that is exporting a view and the technology stack and consensus mechanism it is built on [Naka08][Kiay16][Kiay17][PoS][PoET][PoA].

9.2. Temporal guarantees of view authenticity

The view address as specified in this draft has no temporal component, implicitly conveying a request for the latest or “freshest” state projection from a shared ledger. Even apart from expanding the specification of the view address to include, for example, an absolute or relative timestamp, we will need to augment the data sharing protocol with a mechanism to convey proof of temporal veracity of a view. Work done on verifiable observation of shared ledgers using a public bulletin board [Abebe21] can be taken as the starting point for such a specification.

10. References

10.1. Normative References

[FATF] FATF, "International Standards on Combating Money Laundering and the Financing of Terrorism & Proliferation - The FATF Recommendations", October 2018, <http://www.fatf-gafi.org/publications/fatfrecommendations/documents/fatf-recommendations.html>.

[ISO] ISO, "Blockchain and distributed ledger technologies- Vocabulary (ISO:22739:2024)", January 2024, <https://www.iso.org/standard/82208.html>.

[NIST] Yaga, D., Mell, P., Roby, N., and K. Scarfone, "NIST Blockchain Technology Overview (NISTR-8202)", October 2018, <https://doi.org/10.6028/NIST.IR.8202>.

[RFC1630] Berners-Lee, T., "Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", RFC 1630, DOI 10.17487/RFC1630, June 1994, <https://www.rfc-editor.org/rfc/rfc1630>.

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, DOI 10.17487/RFC1738, December 1994, <https://www.rfc-editor.org/rfc/rfc1738>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <https://www.rfc-editor.org/rfc/rfc2616>.

[SATA] Hardjono, T., Hargreaves, M., Smith, N., and V. Ramakrishna, "Secure Asset Transfer (SAT) Interoperability Architecture, IETF, draft-ietf-satp-architecture-09", February 2026, <https://datatracker.ietf.org/doc/draft-ietf-satp-architecture/>.

[SATP] Hargreaves, M., Hardjono, T., Belchior, R., Ramakrishna, V., and A. Chiriac, "Secure Asset Transfer Protocol (SATP) Core, IETF, draft-ietf-satp-core-13", March 2026, <https://datatracker.ietf.org/doc/draft-ietf-satp-core/>.

10.2. Informative References

[Abebe21] Abebe, E., Hu, Y., Irvin, A., Karunamoorthy, D., Pandit, V., Ramakrishna, V., and J. Yu, "Verifiable Observation of Permissioned Ledgers (ICBC 2021)", May 2021, <https://arxiv.org/abs/2012.07339>.

[Andr18] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolic, M., Weed Cocco, S., and J. Yellick, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains, EuroSys", January 2018, <https://arxiv.org/abs/1801.10228>.

[BVGC20] Belchior, R., Vasconcelos, A., Guerreiro, S., and M. Correia, "A Survey on Blockchain Interoperability: Past, Present, and Future Trends", May 2020, <https://arxiv.org/abs/2005.14282v2>.

[Clar88] Clark, D., "The Design Philosophy of the DARPA Internet Protocols, ACM Computer Communication Review, Proc SIGCOMM 88, vol. 18, no. 4, pp. 106-114", August 1988.

[Ethe22] "Ethereum whitepaper", September 2022, <https://ethereum.org/en/whitepaper/>.

[Gray81] Gray, J., "The Transaction Concept: Virtues and Limitations, in VLDB Proceedings of the 7th International Conference, Cannes, France, September 1981, pp. 144-154", September 1981.

[Harer2022] Hrer, F., "Towards Interoperability of Open and Permissionless Blockchains: A Cross-Chain Query Language, IEEE International Conference on e-Business Engineering

(ICEBE), Bournemouth, United Kingdom, 2022, pp. 190-197, doi: 10.1109/ICEBE55470.2022.00041", October 2022, <https://ieeexplore.ieee.org/document/10035048>.

[Hear19] Hearn, M. and R. G. Brown, "Corda: A Distributed Ledger", August 2019, <https://docs.r3.com/en/pdf/corda-technical-whitepaper.pdf>.

[Herl19] Herlihy, M., "Blockchains from a Distributed Computing Perspective, Communications of the ACM, vol. 62, no. 2, pp. 78-85", February 2019, <https://doi.org/10.1145/3209623>.

[HLP19] Hardjono, T., Lipton, A., and A. Pentland, "Towards an Interoperability Architecture for Blockchain Autonomous Systems, IEEE Transactions on Engineering Management", June 2019, <https://ieeexplore.ieee.org/document/8743548>.

[HS2019] Hardjono, T. and N. Smith, "Decentralized Trusted Computing Base for Blockchain Infrastructure Security, Frontiers Journal, Special Issue on Blockchain Technology, Vol. 2, No. 24", December 2019, <https://doi.org/10.3389/fbloc.2019.00024>.

[Kiay16] Kiayias, A., Lamprou, N., and A. Stouka, "Proofs of proofs of work with sublinear complexity, International Conference on Financial Cryptography and Data Security, pages 6178, Springer", 2016.

[Kiay17] Kiayias, A., Miller, A., and D. Zindros, "Non-Interactive Proofs of Proof-of-Work, Cryptology ePrint Archive, Paper 2017/963", September 2017, <https://eprint.iacr.org/2017/963>.

[Naka08] Nakamoto, S., "Bitcoin: A Peer-to-Peer Electronic Cash System, Decentralized Business Review", October 2008, <https://bitcoin.org/bitcoin.pdf>.

[PoA] Antolin, M., "What Is Proof-of-Authority? Understanding PoA Consensus Mechanisms, CoinDesk", June 2022, <https://www.coindesk.com/learn/what-is-proof-of-authority/>.

[PoET] Bowman, M., Das, D., Mandal, A., and H. Montgomery, "On Elapsed Time Consensus Protocols, Cryptology ePrint Archive, Paper 2021/086", 2021, <https://eprint.iacr.org/2021/086>.

[PoS] "Proof-of-Stake (PoS) | ethereum.org", September 2022, <https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/>.

[SRC84] Saltzer, J., Reed, D., and D. Clark, "End-to-End Arguments in System Design, ACM Transactions on Computer Systems, vol. 2, no. 4, pp. 277-288", November 1984.

Authors' Addresses

Venkatraman Ramakrishna
IBM Research
Email: vramakr2@in.ibm.com

Vinayaka Pandit
IBM Research
Email: pvinayak@in.ibm.com

Ermyas Abebe
Consensys
Email: ermyas.abebe@consensys.net

Sandeep Nishad
IBM Research
Email: sandeep.nishad1@ibm.com

Krishnasuri Narayanam
IBM Research
Email: knaraya3@in.ibm.com