

Offline-verifiable Data from Distributed Ledger-based Registries

Stefan More^a, Jakob Heher and Clemens Walluschek
Institute of Applied Information Processing and Communications (IAIK),
Graz University of Technology, Graz, Austria

Keywords: Distributed Ledger, Trust Management, Offline Verification, Privacy.

Abstract: Trust management systems often use registries to authenticate data, or form trust decisions. Examples are revocation registries and trust status lists. By introducing distributed ledgers (DLs), it is also possible to create decentralized registries. A verifier then queries a node of the respective ledger, e.g., to retrieve trust status information during the verification of a credential. While this ensures trustworthy information, the process requires the verifier to be online and the ledger node available. Additionally, the connection from the verifier to the registry poses a privacy issue, as it leaks information about the user's behavior. In this paper, we resolve these issues by extending existing ledger APIs to support results that are trustworthy even in an offline setting. We do this by introducing attestations of the ledger's state, issued by ledger nodes, aggregatable into a collective attestation by all nodes. This attestation enables a user to prove the provenance of DL-based data to an offline verifier. Our approach is generic. So once deployed it serves as a basis for any use case with an offline verifier. We also provide an implementation for the Ethereum stack and evaluate it, demonstrating the practicability of our approach.

1 INTRODUCTION

A trust management system answers trust questions by executing a policy defined by its operator. Such a policy specifies what credentials a user needs to provide to access a resource, and which rules a credential needs to fulfill to be accepted as “trusted” by the system. The policy is evaluated with regard to a set of trust information coming from different sources. On one side, the *user* (prover) provides their own credentials as input to the system. On the other side, the *verifier* often requires more information to authenticate those credentials, such as their revocation status or data about the trustworthiness of their issuer (Alber et al., 2021; Mödersheim et al., 2019). Since the verifier needs to trust this additional information, it is typically collected directly from respective authorities and registries.


A distributed ledger (DL) is an attractive technology to maintain such registries. A classic example is the common use case of revocation: Since credentials expire and mistakes happen, issuers want to be able to revoke a credential. In the Self-Sovereign Identity (SSI) world, this is often realized using a revocation registry in form of a list stored on a DL. Other exam-

ples are projects that use a DL to establish a Web of Trust as a distributed trust store and storage for credential schema information (FutureTrust Consortium, 2020; More et al., 2021).

Challenge: Availability & Privacy. To retrieve DL-based data, the verifier communicates with the API of a DL node it trusts. While this ensures the freshness of the data, a network connection to this node is required. If the verifier is offline, it cannot reach the DL, and thus cannot retrieve a trustworthy copy of the data (Abraham et al., 2020). The same is true if the DL node used by the verifier is unavailable (Li et al., 2021).

An additional challenge is the privacy of users since such approaches don't allow *unobservability* of interactions. The issue is that the contacted DL node learns about the showing of a credential, and about the verifier the credential was shown to. Since verifiers are typically operated by the individual service provider, this correlates with the user's associations (Chung et al., 2018). Often, sensitive information such as physical location can be derived.

As data provided by the DL API is currently not signed, trust in it is derived solely from the authenticity of the underlying connection with the

^a  <https://orcid.org/0000-0001-7076-7563>

trusted node. This is in contrast with comparable technologies, such as OCSP stapling in TLS (Eastlake, 2011).

Contribution. In this paper, we solve the described problem with a generic *Ledger State Attestation (LSA)* system (cf. Section 2). Using this LSA system, a user can retrieve data from a DL and prove to an offline verifier the provenance of this data. Since our approach provides a generic interface to the data stored on the ledger, the system can be used in different use cases.

(I) **Node Attestations:** We enable DL nodes to issue signed *node attestations* to users (cf. Section 2.1). Such an attestation contains the result of some specified operation on the DL, such as retrieving the current block hash or the result of a smart contract invocation. Additionally, it attests in an offline-verifiable way that the result matches the node's current view of the DL state. We achieve this without modifications to the code of the ledger clients but instead provide a wrapper around the node API. This approach is also transparent to the consensus protocol used by the ledger. The wrapper provides a generic attestation functionality and thereby supports all kinds of current and future use cases. Although this wrapper needs to be hosted directly on the nodes' servers, this only needs to be done once.

(II) **Aggregate Attestations:** We enable an user to retrieve such node attestations from multiple nodes aggregated into a single *aggregate attestation* (cf. Section 2.1). By retrieving node attestations from an appropriate set of DL nodes, the user can be reasonably sure that the aggregate attestation also includes node(s) that an unknown verifier trusts. The verifier can then verify the attestation without needing to communicate with the node(s) in question. As it can now trust the provided result, it can then use it to authenticate the user's credentials while remaining fully offline and without leaking information to the node or other third parties.

(III) **Implementations:** We demonstrate the feasibility of our approach using two implementations (cf. Section 3). While our general architecture is independent of a concrete DL technology, in this proof of concept implementation we focus on the Ethereum stack. Our first variant enables DL nodes to attest the current block hash, which then allows an offline verifier to establish trust in any DL-based data that the user provides. The second variant issues attestations of returned data from smart contract function calls. This enables users to specify custom queries or filters for the data they want to retrieve.

1.1 Related Work

Various systems and methods in the blockchain world use data stored in a DL, but all of those systems have online components that directly interact with the ledger. E.g., Layer 2 protocols (Gudgeon et al., 2020) move a large number of transactions from a ledger to an off-chain service. Systems based on the layer 2 approach interact with a smart contract and thus require a connection to the DL. The same requirement exists for Ethereum's Light Client, which fetches a state root from a trusted node (Chatzigiannis et al., 2021). Another example is inter-ledger communication (Zamyatin et al., 2021), which is used to transfer assets from one ledger to another. This transfer requires a trusted third party with a connection to both ledgers.

To prove the provenance of data, TLS-N¹ uses a more generic approach by extending the TLS handshake to enable a server to notarize a TLS session. TLSNotary² and DECO (Zhang et al., 2020) are concerned with the attestation of access protected web data to a third party. This is realized by involving a third party (oracle) trusted by the verifier in the TLS session with the server.

Some revocation systems work without a direct connection between the verifier and the revocation authority. One common example of this is the verification of TLS certificates using OCSP stapling (Eastlake, 2011). The system by Abraham et al. (Abraham et al., 2020) allows the offline verification of DL-based revocation information. However, it is limited to only the revocation use case. Modifying this system for other use cases is possible, but such modifications require adaptations on all nodes of a DL for each additional use case.

1.2 Background

Distributed Ledger (DL). A DL is a redundant append-only datastore on distributed nodes without central control (Jannes et al., 2019; Alexopoulos et al., 2017). The nodes agree on the ledger's current state by running a consensus protocol (Xiao et al., 2020). The distribution of nodes can be geographical, political, institutional, etc. to prevent collusion and improve resilience. DLs can be classified into different access models, depending on who can join the network (public vs. private), or by who has read and write access to it (permissioned vs. permissionless). In this work we focus on permissioned ledgers.

¹<https://github.com/tls-n>

²<https://tlsnotary.org/TLSNotary.pdf>

Smart Contract (SC). Many ledgers support the storing of code on the DL, which is then deterministically executed by the nodes performing the consensus protocol. Examples of this are the Ethereum ledger³ and various ledgers from the Hyperledger project.⁴ Another example is Hyperledger Besu,⁵ a Ethereum client specifically designed for use in permissioned consortium ledgers. Variables in the code are stored on the DL as well and can be read and modified using functions supplied by the contract. SC code is written in a high-level language and then compiled to ledger-specific bytecode, which is then written to the DL. When a user sends a function call to a contract, nodes execute this bytecode, for example using the Ethereum Virtual Machine (EVM).⁶ The resulting state is only written to the ledger if all nodes agree on the result of the computation.

BLS (Multi-)Signatures. Boneh–Lynn–Shacham (BLS) is a provable secure pairing-based signature scheme producing short signatures (Boneh et al., 2004). One property of BLS is that it can be used as a multi-signature scheme: signatures issued by multiple private keys can be aggregated into a single constant-size signature, saving space and verification time (Boldyreva, 2003; Boneh et al., 2018).

2 DESIGN

There are several main components in our system. We describe them in the following. A high-level and generic overview of these components and how they are connected is shown in Figure 1, while Figure 2 gives a more detailed and Ethereum-focused overview.



Figure 1: High-level architecture of our Ledger State Attestation system.

The distributed ledger (DL) is represented by its **DL Nodes**. These nodes communicate peer-to-peer, using a consensus protocol to agree on a shared state. DL Nodes provide an HTTP API to allow other entities to access the state of the DL.

We add the **LSA Wrapper** component to a DL node. It wraps the node API, providing access to the

same data but enriches the API functionality, adding a proof of provenance to the returned data. The wrapper provides the same API endpoints as the node API, so it is compatible with existing API clients.

We introduce the **LSA Gateway** stand-alone component to support the user by retrieving data from all DL nodes. It can be part of a node, run by the user, or be operated by a third party. We discuss the implications of this choice in more detail in Section 4. The gateway provides the same API endpoints as the node API and our LSA Wrapper. It forwards queries it retrieves from a user to the DL nodes, and aggregates their answers into a single response for the user.

The **User** wants to retrieve data from a DL, and present it to a verifier. To retrieve data directly from the DL, the users interact with the API of a DL node. For data that can be presented to an offline verifier later, they instead interact with the LSA Gateway.

The **(Offline) Verifier** receives data from the user and needs to establish trust in this data. We consider a scenario where this verifier is offline, i.e., it cannot connect to any of the DL's nodes during verification.

In DL-based trust management, the verifier is interested in the trustworthiness of a **claim** about the DL's state. This claim can, e.g., be an assertion of the current block hash, or of the return value of a smart contract function. We define an **Attestation** as data combined with proof of authenticity. There are two types of attestations: A **node attestation**, created by an individual DL node, attests that the data reflects the data in its local storage. Combining such attestations of several nodes yields an **aggregate attestation**, attesting an agreement between all involved nodes. If an aggregate attestation contains the attestations of *all* nodes of a DL, we call this a **ledger attestation**.

2.1 Ledger State Attestation (LSA)

When a user shows some data to a verifier, this verifier needs to make sure that it can trust this data before relying on it for further processing. Trusting data coming from a DL-based registry means verifying that this data matches the data stored in the DL. An online verifier could check this by contacting a trustworthy DL node and comparing the data, or even by doing additional lookups on its own.

Since we consider the scenario where a verifier is offline, this online check is not possible. The underlying challenge is that an offline verifier has no reason to trust data that a user claims to have retrieved directly from a node's API. For example, a verifier cannot be sure if this data was really retrieved from a DL, that the user did not alter the data, or that the data represents the latest state of the DL.

³<https://docs.soliditylang.org>

⁴<https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode4ade.html>

⁵<https://besu.hyperledger.org>

⁶<https://ethereum.org/en/developers/docs/evm>

Attestation of State by a Node. To mitigate this problem, we add the LSA Wrapper to all nodes of the DL, wrapping the node API. This is the only modification to a DL node our approach requires, and it only needs to be done once and not for every use case. While the default node API answers user queries by returning plain data, the wrapper additionally adds a proof to the response. To ensure the authenticity of the data, the wrapper creates this attestation proof using a private key of the node. To enable a verifier to decide if the presented information was fresh enough, the number of the current block and a low-resolution timestamp are also added to the attestation.

The attestation can then later be presented to an offline verifier which checks it to ensure that the presented data was returned by a specific DL node and has not been altered before processing the data.

Attestation by the Whole DL. While this process ensures authenticity (and integrity) of the data with respect to one node, it means the user needs to select a node the verifier trusts. This is a problem since a user does not know, at the time of retrieving the attestation, which node(s) a verifier trusts. Additionally, this limits which verifier the user can present an attestation to, since different verifiers trust different nodes. To avoid this, the user would need to retrieve an attestation by all DL nodes.

In our LSA system, the gateway is used to provide users with an easy way to retrieve data, alongside a proof from all nodes. Since the data stored by each node was agreed upon using the consensus protocol, the data returned is also the same for each node. But the proof returned by the nodes is different since it proves authenticity for a certain node. This allows that the gateway returns the data only once, and aggregates the proofs of the nodes into a single proof.

3 IMPLEMENTATION

To show the feasibility of our LSA concept, we implement a prototype for the Ethereum stack. We focus on a permissioned DL using Hyperledger Besu.

On the server-side, we provide a wrapper for Ethereum's RPC API, which extends the API of Ethereum nodes to support signed responses. The LSA Gateway first forwards the user's query to the wrapped API of all applicable nodes. Then, it aggregates the received node attestations into one aggregate attestation. For the authenticity proofs, we use digital signatures issued individually by each node using their private key. To be able to aggregate the signature of the individual nodes into one combined signature,

we use the BLS signature scheme for the authenticity proofs (Boneh et al., 2020). An additional benefit of using BLS is its efficiency and small storage requirements, minimizing the overhead (cf. Section 3.1).

On the client side, we extend the *web3.js* library⁷ to retrieve node and aggregate attestations. This allows a user to call contracts using well-established high-level calls, but retrieve the response value in a signed and offline-verifiable form.

To demonstrate the flexibility of our design, we implement two different variants, which differ in the type of data that gets attested.

Variant 1: Attestation of Data. In this variant we enable an offline verifier to trust any raw data retrieved from the DL by the user. This then allows the verifier to execute a locally stored smart contract or work with the data by other means.

Ethereum ledgers protect the integrity of their data using merkle trees: The block hash is the root hash of a merkle tree, formed by all transactions, smart contract code, and data stored in the ledger at a certain point in time (Wood et al., 2022). A trustworthy attestation of the block hash therefore allows any data in that block, and any previous block, to be trusted. Another advantage of the attestation of the root hash is that it can be pre-computed. Since this needs to be done only once per block, this significantly reduces the load on the DL nodes while still allowing to establish trust in all data on the DL.

Thus, we create a mechanism to retrieve an attestation of the current root hash. This allows a user to retrieve any raw data, and the supplementary parts of the merkle tree, called merkle proof⁸, from any (single) node. Afterward, they send this data, the merkle proof, and the attestation of the root hash to the verifier. The verifier can then use this trusted hash to establish trust in the data.

Variant 2: Attestation of Smart Contract Response. In our second variant, we enable users to query for and retrieve trustworthy data from the DL. To do so, we extend the functionality of the node API in a way that nodes can issue attestations for the return value of smart contract functions. Part of this attestation is also the address of the called smart contract, the executed function, and the call parameters. This enables users to send queries to a smart contract and get an attestation of the query result. To provide more flexibility, we do this by extending the `call` function

⁷<https://github.com/ethereum/web3.js>

⁸<https://eips.ethereum.org/EIPS/eip-1186>

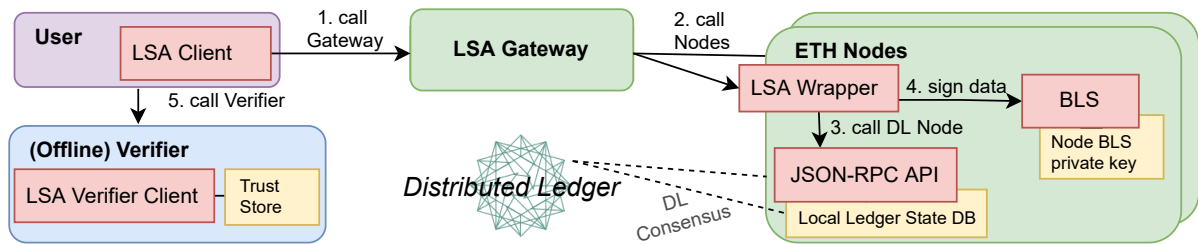


Figure 2: The architecture of our Ledger State Attestation system, extending the functionality of Ethereum nodes and web3.js.

of the web3 client library with the ability to request and handle signed attestations.

A user can simply execute the function in the same way as without the LSA system. The result is an aggregate attestation that contains the call and return value of a certain contract function, and proves the consensus of the nodes about this state. This attestation credential can be shown to an offline verifier and authenticated using the verifier’s truststore. Since the attestation contains the needed data, the verifier does not need to execute a smart contract or evaluate a merkle proof.

3.1 Evaluation

We consider the performance of our LSA approach. To do this, we contrast it with traditional online verification. We identify the following additional costs.

Attestation Retrieval requires an additional network round trip compared to a traditional online query, in scenarios where the LSA Gateway is not co-located on the user device. Quantifying this overhead exactly is difficult, as it depends on the devices’ physical location. Given a transatlantic round trip time of around 90 ms⁹, we consider this to be negligible.

Data Attestation requires each node to create a signature over the data retrieved from the DL. Signature creation takes ≈ 0.3 ms on a typical consumer laptop (Boneh et al., 2020).

During verification the user needs to **transmit the retrieved LSA** to the verifier. In BLS, both signature and public key are encoded as single group elements (Boneh et al., 2020). Thus, an aggregated signature uses 48 bytes, with an additional 48 bytes per public key. For a DL with 20 nodes, transmitting the 1 kB of signature metadata alongside 10 kB of data only takes ≈ 150 ms, even using Bluetooth 4.2.

Then, the verifier needs to **verify the LSA’s signature**. For BLS signatures with 128-bit security, this takes roughly 2.7 ms on a typical consumer laptop (Boneh et al., 2020).

We note that transmission time, scaling with the size of the transmitted data and number of involved nodes, appears to be the primary driver of verification time. This presents potential optimizations by reducing the size of the transmitted LSA. Regardless, we consider a total duration overhead of ≈ 153 ms to be negligible for an interactive showing (Nielsen, 1997).

4 DISCUSSION

4.1 Trust Assumptions

The User must trust the verifier’s trusted DL nodes to provide truthful attestations. This assumption is also made in the online case, and is not unique to our work.

Additionally, heading into an offline scenario the user relies on the provided attestation being valid. It is not a negligible concern that the LSA Gateway returns a bogus attestation. In order for the user to verify the provided attestation, they would need to have a list of all DL nodes’ keys on their device. In general, this is not trivial (see also Section 4.2). Therefore, the user must trust the LSA Gateway to provide a valid attestation. They may also retrieve attestations from multiple different LSA Gateways. As long as at least one returns a valid attestation, the user device successfully complete the LSA process.

The Verifier has some trust policy that relies on the truthfulness of some subset of DL nodes. To verify the attestation proof, the verifier also has a store with the public keys of the nodes it trusts. This does not require additional assumptions beyond those already made in the online scenario. The LSA Gateway simply retrieves attestations from all DL nodes. The verifier’s trust in the aggregate attestation derives from the inclusion of attestations by nodes it trusts. It does not need to trust the LSA Gateway. Indeed, the existence of the gateway is transparent to the verifier.

⁹<https://enterprise.verizon.com/terms/latency/>

4.2 Operational Concerns

Availability. If the verifier expects an attestation was signed by *all* nodes of a certain trust subset, and the LSA Gateway was not able to reach all of these nodes, the resulting attestation will be rejected.

To mitigate this, verifiers could use a threshold policy. Using the list of public keys that are part of an attestation, the verifier first verifies the aggregated signature on the data. It then checks if at least k nodes from its trust store signed the provided aggregate, and accepts it if so.

Required Modifications. Modifications to existing systems are always a challenge, especially to nodes in a distributed system. An advantage of our approach is that the only such modification is the addition of the LSA Wrapper to the nodes, which provides generic attestation and can thus be employed in various use cases. Such a modification could be for example performed during the setup of the system, and only the nodes considered by any verifier need to be modified. This is in contrast to the state of the art, where each use case requires an additional modification to the DL nodes, which is often not feasible during operation.

5 CONCLUSIONS

In many previous decentralized trust systems, an implicit always-online requirement is a major hindrance to practical applicability. We resolve this issue by applying the battle-tested concept of OCSP stapling to the distributed ledger ecosystem.

In this work, we introduced Ledger State Attestations, which allow arbitrary queries to DL nodes' HTTP API to retrieve attested results. This serves as the basis for almost any imaginable use case with only a single adjustment to the underlying DL's nodes, and is a significant improvement over the state of the art. Additionally, our LSA approach enables unobservability of interactions with the verifier, which is an important property to ensure the privacy of users.

Furthermore, we provided a proof of concept implementation for Ethereum-based ledgers. We evaluate this implementation, demonstrating the practical feasibility of our scheme.

ACKNOWLEDGEMENTS

This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement № 871473 (KRAKEN).

REFERENCES

- Abraham, A., More, S., Rabensteiner, C., and Hörandner, F. (2020). Revocable and offline-verifiable self-sovereign identities. In *TrustCom*. IEEE.
- Alber, L., More, S., Mödersheim, S., and Schlichtkrull, A. (2021). Adapting the TPL trust policy language for a self-sovereign identity world. In *Open Identity Summit*, LNI. Gesellschaft für Informatik e.V.
- Alexopoulos, N., Daubert, J., Mühlhäuser, M., and Habib, S. M. (2017). Beyond the hype: On using blockchains in trust management for authentication. In *TrustCom*. IEEE.
- Boldyreva, A. (2003). Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC*, LNCS. Springer.
- Boneh, D., Drijvers, M., and Neven, G. (2018). Compact multi-signatures for smaller blockchains. In *ASIACRYPT*, LNCS. Springer.
- Boneh, D., Gorbunov, S., Wahby, R. S., Wee, H., and Zhang, Z. (2020). BLS Signatures. Internet-Draft draft-irtf-cfrg-bls-signature-04, Internet Engineering Task Force. Work in Progress.
- Boneh, D., Lynn, B., and Shacham, H. (2004). Short signatures from the weil pairing. *J. Cryptol.*
- Chatzigiannis, P., Baldimtsi, F., and Chalkias, K. (2021). Sok: Blockchain light clients. *IACR Cryptol. ePrint Arch.*
- Chung, T., Lok, J., Chandrasekaran, B., Choffnes, D. R., Levin, D., Maggs, B. M., Mislove, A., Rula, J. P., Sullivan, N., and Wilson, C. (2018). Is the web ready for OCSP must-staple? In *IMC*. ACM.
- Eastlake, D. (2011). Transport layer security (tls) extensions: Extension definitions. RFC 6066, RFC Editor.
- FutureTrust Consortium (2020). Global Trust Service List. <https://pilots.futuretrust.eu/gtsl>. online, accessed on 22 January 2022.
- Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., and Gervais, A. (2020). Sok: Layer-two blockchain protocols. In *Financial Cryptography*, LNCS. Springer.
- Jannes, K., Lagaisse, B., and Joosen, W. (2019). You don't need a ledger: Lightweight decentralized consensus between mobile web clients. In *SE-RIAL@Middleware*. ACM.
- Li, K., Chen, J., Liu, X., Tang, Y. R., Wang, X., and Luo, X. (2021). As strong as its weakest link: How to break blockchain dapps at RPC service. In *NDSS*. The Internet Society.
- Mödersheim, S., Schlichtkrull, A., Wagner, G., More, S., and Alber, L. (2019). TPL: A trust policy language. In *IFIPTM*. Springer.
- More, S., Grassberger, P., Hörandner, F., Abraham, A., and Klausner, L. D. (2021). Trust me if you can: Trusted transformation between (JSON) schemas to support global authentication of education credentials. In *IFIP SEC*. Springer.
- Nielsen, J. (1997). Usability engineering. In *The Computer Science and Engineering Handbook*. CRC Press.

- Wood, G. et al. (2022). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper, Berlin version*.
- Xiao, Y., Zhang, N., Lou, W., and Hou, Y. T. (2020). A survey of distributed consensus protocols for blockchain networks. *IEEE Commun. Surv. Tutorials*.
- Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sanchez, P., Kiayias, A., and Knottenbelt, W. J. (2021). Sok: Communication across distributed ledgers. In *Financial Cryptography*, LNCS. Springer.
- Zhang, F., Maram, D., Malvai, H., Goldfeder, S., and Juels, A. (2020). DECO: liberating web data using decentralized oracles for TLS. In *CCS*. ACM.

