REVISION-SAFE ARCHIVING AND LICENSE-CONTROLLED ACCESS USING DISTRIBUTED LEDGER TECHNOLOGY

Sven Schlarb
AIT Austrian Institute of Technology
Austria
sven.schlarb@ait.ac.at
0000-0003-3717-0014

Roman Karl
AIT Austrian Institute of Technology
Austria
roman.karl@ait.ac.at

Victor-Jan Vos
NIOD Instituut voor Oorlogs-, Holocaust- en Genocidestudies
Netherlands
v.vos@niod.knaw.nl

Carlijn Keijzer
NIOD Instituut voor Oorlogs-, Holocaust- en Genocidestudies
Netherlands
c.keijzer@niod.knaw.nl

Begoña Sanchez Royo
Highbury Research and Development Ltd.
Ireland
begona.sanchez-royo@highbury.ie
0000-0002-2911-7881

Abstract – This paper describes an approach and a prototype system to make use of Distributed Ledger Technology or, more specifically, Blockchain, to build a trusted digital repository with a transparent and traceable change record for events related to the preservation or action of requesting or granting access to digital information objects. The approach focuses on a notary use case where the information stored in the blockchain serves as a proof of evidence regarding the existence and integrity of digital information objects.

Keywords – Blockchain, Distributed Ledger, Digital Repository, Electronic Archiving.

Conference Topics – Digital Accessibility, From Theory to Practice

1. INTRODUCTION

In this paper we present an approach together with a prototype implementation [1] which aims at increasing trust in electronic archiving and digital repositories by enabling a transparent and traceable change history of archival records using distributed ledger technology (DLT) or Blockchain systems.

Digital objects stored in a repository are subject to a life cycle, that is, there are events that modify the objects themselves or the metadata related to them. Trustworthy archiving means that these changes are recorded as events in a transparent manner and that it is clearly documented who initiated the changes and for what reason.

One of the well-known application domains of Blockchain is the so called “decentralized notary” [2, section 7]. The principle is that the piece of data, such as the fingerprint (or hash value) – demonstrating the existence and integrity of a document – is stored in the Blockchain together with a timestamp. It is decentralized because – depending on the security setup – any node can initiate transactions which are then processed through the distributed consensus and added to the Blockchain.

We present a use case related to negotiating access to digital objects where an applicant – a researcher from the repository’s designated community or a general user – requests access. Further we describe

iPRES 2023: The 19th International Conference on Digital Preservation, Champaign-Urbana, IL, US.

https://hdl.handle.net/2142/121115

that the basic principle can also be used to record digital preservation events.

The event metadata can be persisted together with the information resource. We use PREMIS, a widely used metadata scheme for recording preservation events, in combination with blockchain transactions to provide a transparent, auditable and tamper proof change history record.

The prototype implementation [1] demonstrates the principle of using a blockchain notary for recording events related to accessing or preserving digital objects and is designed to make use of the use of the European Blockchain Infrastructure Service (EBSI) [3] which allows making use of API functions to build a transparent and tamper-proof provenance and change history record without the need to set up a dedicated blockchain service infrastructure. The European Blockchain Services Infrastructure (EBSI) is a cooperation of 29 countries and the European Commission. It is a private blockchain-based system with about 30 nodes which largely builds upon the Ethereum ecosystem with several smart contracts written in Solidity defining the core of the EBSI functionality. This private Ethereum network is not accessible directly from outside by users and external developers. Instead, there are several higher-level APIs as the only way to access the system from outside. Developers can use this API to write decentralized applications in a similar way as with interacting with custom smart contracts directly, but with the additional EBSI compatibility.

The paper's outline is structured as follows: Section 2 provides an overview of related initiatives and work. Section 3 elaborates on the fundamental methods for interacting with the blockchain. Section 4 delves into the implementation details of the access and preservation use cases. Finally, Section 5 concludes the paper by summarizing the key findings.

2. Related Initiatives and Work

A series of standards is relevant for trustworthy archiving: The Reference Model for an Open Archival Information System (OAIS) defines the requirements for an archive or repository to provide long-term preservation of digital information. Based on the reference model several initiatives produced recommendations for certification criteria related to trustworthy repositories, such as [4], [5], and [6]. Even though these publications address mainly organizational infrastructure aspects and do not address the technical means for building trust, they represent the general frame for building “accountable record-keeping systems” [5, p. 8].

The relevance of blockchain technology for archiving is reflected in the large number of publications related to this topic. Very close to the approach presented here is the model of a blockchain-based system to assist the process of long-term preservation of digitally signed records presented in [7] and [8] and the project ARCHANGEL [9]. The difference of our approach is that we present a generic use case applicable to any type of archive and propose a way to link preservation and access metadata with the blockchain registry.

3. Interacting with the Blockchain

To be able to interact with the blockchain, or more technically, to send a transaction that will be included in a block, i.e., writing data to the blockchain, an account on the blockchain system is required. This account can correspond either directly to a user or to an external system that administers multiple user accounts by itself. Additional to a digital account protected by public-key cryptography, a link to a person or an organization must be established. We will focus more on persons, “natural persons” in legal terms, as the relevant legal contracts are often concluded on this level.

The proof-of-concept implements a user management that does not include an official verification of an identity by the respective national agencies of EU states. But the architecture will be modular to allow such an extension without major code changes. A potential extension could integrate eID [10], a digital building block that was created as part of the Connecting Europe Facility (CEF) program, which takes care of cross-border verification of identities. Similarly, the European Self-Sovereign Identity Framework (ESSIF) built into EBSI can facilitate the generation of user accounts with verified identities based on official documents and make it usable with the rest of the EBSI functionality.

Giving access to data is also not a task that is typically solved only by a smart contract. First, because most blockchain systems have no sophisticated reading protection, which means that data is readable by default for parties with access to the system. But a blockchain is often not an ideal storage system for many kinds of data, because of its persistent nature

and its reduced capacity and throughput. Therefore, dissemination information objects will clearly not be stored on the blockchain. Enforcing the rules of the blockchain will be done by a component outside the blockchain system. This component does not need a distributed architecture and can be located at the archive. For multiple archives the component can simply be run as multiple instances, or an archive could use its own implementation if it wishes so. It is important to note that by doing so, we do not lose the advantages of the blockchain system. If an archive gave access to a DIP without having a legal contract established on the blockchain, this would be in its own control and responsibility. On the other hand, an archive that does not give access to an applicant even though a legal contract has been established would violate that agreement, which could be proven by the applicant.

Regarding the roles which are involved in the information access use case we define three entities when establishing a license agreement: Provider Entity (PREMIS metadata: agent), Applicant Entity (PREMIS metadata: agent), and Object Entity (PREMIS metadata: Information Package, Representation, or File).

The license agreement is concluded between the Provider Entity and the Applicant Entity, and it relates to the Object Entity.

For the preservation use case, events, such as the migration of representations of file objects, are documented as PREMIS with the corresponding agents documenting the preservation decision taken. The notary function of the blockchain registers a combined hash of event identifier and representation or file hash in this case.

4. Use Case Implementation

One of EBSI's APIs is particularly interesting for referencing data entries to prove their existence and validity at a certain point in time. The so called "Timestamp API" basically stores hashes and associates them to the timestamps at the point of time when the entry was created. The hashing algorithm can be chosen by the user out of a list of standardized algorithms.

The timestamp is added by the EBSI system when the entry is written to the blockchain. The original data can be stored off-chain. That might be possible also via the EBSI infrastructure or outside of it via a separate application.

An important aspect of the dissemination of digital objects is the definition of rights and the agreement concerning the usage. To define in which way and to what extent a dissemination object can be used, a legally binding contract between two parties, the provider, and a requester, must be put in place. In the context of systems that use a blockchain as basic data structure, there are pieces of code, called smart contracts, which are sometimes seen as an automated form of a legal contract. But this is only true in certain cases because smart contracts can only control very specific aspects of a legal contract. A smart contract cannot prevent the requester to use the dissemination object in any way that would violate the legal contract. But still, having a blockchain as data structure where it is not possible to modify existing entries helps us to digitally record an agreement between two parties and put the legally binding contract in place. The central part of the legal contract is the text that describes the legal aspects, which we call "license".

The process of recording the agreement on the blockchain consists of five steps:

1. Register dissemination representation for access. The dissemination object is identified and referenced to by a UUID. The dissemination object itself is never put on the blockchain, but only its identifier.
2. Create text license document. Most of the time, we will deal with a small set of standard licenses, but it is also possible to assemble a license out of a set of standard clauses or to set up an individual license. A license is hashed to prohibit future modifications and the hash is used as identifier for a license on the blockchain. Like the dissemination object, the license itself won't be stored on the blockchain.
3. Provider assigns license to dissemination representation. An entry on the blockchain is made to record the bundling of the dissemination object with a specific license. Everyone with access to the system will then be able to see the available offers. Since the blockchain contains only the identifiers, a customer will need not only the information from the blockchain to assess the offer. What

is important, is that the data on the blockchain is sufficient for a customer to verify the validity of the offer.

4. Requester accepts license. The requester is the first to sign the offer via an entry on the blockchain.
5. Provider approves request. The signature of the provider via an entry on the blockchain finalizes the legally binding contract between one requester and provider for one dissemination object.

In these five steps we have three different fields identifying the different objects and the user (field: object identifier, type UUID, size: 16 bytes; field: license hash, type SHA3-256, 32 bytes; field: requester, type: Ethereum address, size: 20 bytes).

In steps 3 to 5, we need to store records which consist of a combination of the fields. If we want to use the Timestamp API, we cannot store multiple fields, but just a single hash value. We compute such a hash value by appending the input bytes in binary format and by then hashing this byte sequence with SHA3-256. As a result, we get another 32 bytes sequence representing a data entry.

This reduction still allows a party to proof the validity of a particular entry at a given time under the condition that the original data is not lost. By itself, the data stored on the blockchain itself will be relatively meaningless, so it is important to see it only as one part of the process.

To demonstrate the approach, we developed a decentralized application that is based directly on Ethereum, but because of the modular design the connection to the blockchain system can be replaced without the need of rewriting code in the other layers of the component. The blockchain-based system is set up as a private network with proof-of-authority as consensus mechanism. Go Ethereum is chosen as software for the execution client and a block is created every 15 seconds. But it is important to note that we do not rely on a particular setup and most of the setup could be changed without affecting the functionality of the decentralized application.

Adhering to the concepts and definitions of the EBSI, we created a timestamp smart contract that resembles the Timestamp service provided by the EBSI but is a simplified implementation. It defines a data structure, a map, with hash values as indexes and timestamps in combination with the address of the creator as associated information. It is implemented in Solidity, which is the most widely used language for smart contracts in Ethereum.

Interacting directly with smart contracts is in general a bit tedious. Most of the time, decentralized applications, sometimes also referred to as *dapps*, are created to interact with smart contracts. In our case the decentralized application is a web server that provides a REST API as an easy way to access the functionality of the smart contract. The web server is written in Haskell, uses GHC 8.10.x, and includes amongst others the libraries *servant* for the REST API and *web3-ethereum* for the connection to the smart contract. To ensure modularity, the application consists of 4 layers, that built only upon the layer directly below, but not on the others.

1. The uppermost layer in the architecture describes the REST API including all elements that concern the web server. This includes the rendering of the responses, for which JSON is used as format.
2. The second layer defines the business logic, which means the five steps in our process of recording an agreement and the functionality for retrieving information.
3. The third layer is responsible for the storage and is specific to the chosen storage backend. In the current implementation, the functions of the custom smart contract are called to store a hash or retrieve the information, timestamp and creator, of a hash. This layer has to be replaced if the EBSI Timestamp API is used instead of the custom smart contract.
4. The lowest layer is also part of the storage and just needed for the direct connection to the Ethereum components and wouldn't be needed with EBSI. It makes all the functionality of the smart contract accessible via the *web3-ethereum* library, as this is a form of polyglot programming and requires some mechanism to connect the different environments.

In the following we briefly outline the REST API functions of the prototype implementation.

- *registerObject*: takes *object_identifier* as parameter and returns the object's registration hash value.
- *createLicense*: takes *license_hash* as parameter and returns the license's registration hash value.
- *assignLicense*: takes *object_identifier* and *license_hash* as parameters and returns the license assignment hash value.
- *acceptLicense*: takes the requester's account requester, the *object_identifier* and *license_hash* as parameters and returns the acceptance hash value.
- *approveRequest*: takes the requester's account requester and the *object_identifier* as parameters and returns the approval hash value.

On the other hand, querying does not involve transactions on the blockchain, and the requests are relatively fast. For any of the registered hash values for object registration, license registration, license assignment, license acceptance, and approval one can get the timestamp of its registration and the associated account address.

- timestamp: takes the registration hash value as parameters and returns the timestamp value timestamp and the creator's account creator.

An Ethereum network is a peer-to-peer network consisting of several nodes. There can be one or more instances of the decentralised application that connect to one node, and one instance can serve one or more users. In the prototype implementation the key store is located at the Ethereum node, but it is possible to relocate it to the decentralised application, which would increase the flexibility and depending on the setup can also help increasing the security. There are some variations depending on the location of the key store and the number of the accounts that are stored in one key store. An interesting case is created by locating the key store at the decentralised application with only one account per key store. This would mean that every user has to run its own web server. An exchange of the custom smart contract with the EBSI services has only a minor impact, as it is also possible with the EBSI to manage the keys either at the node or outside of it.

5. Conclusion

The implementation of the concepts and approach presented in this article can be used with Distributed Ledger Technology or Blockchain services which offer a function for registering a hash value and providing a timestamp as return value. These minimum requirements will allow tracing the creation and integrity of information objects. To also provide evidence for the authenticity of information objects, i.e., who registered them for the first time or who originated specific preservation objects, the events need to be linked to the account. If the requirement is to know about real identities behind the accounts, further identification services, such as the European eID services, need to be integrated.

6. Acknowledgement

This project has received funding from the European Union's CEF programme with action No 2020-EU-IA-0185 under grant agreement No INEA/CEF/ICT/A2020/2397190.

7. References

[1] Kark, R., & Schlarb, S. (2023). Blockchain Notary Proof-of-Concept (Version 1.0.0) [Computer software]. https://doi.org/10.5281/zenodo.8100250

[2] D. Di Francesco Maesa and P. Mori. Blockchain 3.0 applications survey. Journal of Parallel and Distributed Computing, 138:99–114, 2020.

[3] W. J. Buchanan et al., "The Future of Integrated Digital Governance in the EU: EBSI and GLASS." 2023.

[4] D. R. C. T. Force. Trustworthy repositories audit certification: Criteria and checklist. Research libraries group (rlg), Technical report, RLG-NARA, 2007.

[5] R.-O. W. G. on Digital Archive Attributes. Trusted digital repositories: Attributes and responsibilities. Technical report, OCLC, 2002.

[6] W. G. on Digital Archive Attributes. An audit checklist for the certification of trusted digital repositories. RLG-OCLC, 2005.

[7] Bralić, V., Stančić, H. and Stengård, M. (2020), "A blockchain approach to digital archiving: digital signature certification chain preservation", Records Management Journal, Vol. 30 No. 3, pp. 345-362. https://doi.org/10.1108/RMJ-08-2019-0043.

[8] Stančić H, Bralić V. Digital Archives Relying on Blockchain: Overcoming the Limitations of Data Immutability. Computers. 2021; 10(8):91. https://doi.org/10.3390/computers10080091.

[9] J. P. Collomosse et al., "ARCHANGEL: Trusted Archives of Digital Public Documents," CoRR, vol. abs/1804.08342, 2018.

[10] H. Strack, O. Otto, S. Klinner, and A. Schmidt, "eIDAS eID & eSignature based Service Accounts at University environments for cross boarder/domain access." 2019.