IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 24, 2022, accepted February 5, 2022, date of publication February 9, 2022, date of current version February 18, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3150356

Decentralized Attestation and Distribution of Information Using Blockchains and Multi-Protocol Storage

FELIX HÄRER AND HANS-GEORG FILL

Digitalization and Information Systems Group, Department of Informatics, University of Fribourg, 1700 Fribourg, Switzerland

Corresponding author: Felix Härer (felix.haerer@unifr.ch)

This work was supported by the Swiss National Science Foundation Project Domain-Specific Conceptual Modeling for Distributed Ledger Technologies under Grant 196889.

ABSTRACT The distribution of information through web protocols is today based on the client-server model. Recently, decentralized protocols with greater availability appear as well as blockchain-based attestation methods, allowing for proving the existence of information. In combination, these methods promise a secure, decentralized and long-term storage. However, there exist two major problems: (1) the scalability of blockchains limits their storage capacity and (2) various (de)centralized web protocols are in use and could alleviate this problem, but they do not support blockchain-based attestations. In this paper, we extend an approach for blockchain-based attestation with compatibility for multi-protocol storage. Instead of specific protocols or blockchains, the extended approach aims to contribute novel concepts to the discussion on blockchain scalability. It augments the capabilities of existing protocols for applications such as certification or timestamping of digital artifacts. With the use of decentralized protocols such as IPFS, further availability and inherent resilience properties are gained, allowing for applications such as open research repositories and digital registries. We discuss the architecture of the extended approach, a possible implementation in a smart contract on the Ethereum blockchain with IPFS and Git, and evaluate the time and cost of attestations.

INDEX TERMS Blockchain, distributed information systems, decentralized identifiers, distributed file systems, internet protocols.

I. INTRODUCTION

Two central success factors of the traditional world-wide-web were the possibility of identifying any information resource through a *reference* by which they can be retrieved, i.e. the Internationalized Resource Identifiers (IRI), together with the provision of a *network protocol* in the form of HTTP that offered performance and functionalities not available until then [1]. Whereas *integrity* and *privacy* of *transmitted* data have later been enabled through the cryptographic protocols secure-socket-layer (SSL) and transport-layer-security (TLS) [2] for individual transmissions between clients and servers, limitations remain for decentralized storage and availability beyond the client-server model.

These limitations concern, on the one hand, the *integrity* of the storage of data over time and, on the other, their *availability*. As of today, data integrity of references is not verified and preserved, allowing for changes against the intention of the original source. As an example, consider a hyperlink from a website to a scientific dataset used for machine learning. If the referenced dataset changes, e.g. due to an update, the IRI may stay the same. Thus, any agent retrieving the dataset could not infer the integrity of the dataset just from the reference but would have to employ additional measures, e.g. as done today via checksums or digital signatures of the underlying content [3]. The original intention of the creator of the link may thus be violated without actually knowing about the violation.

Secondly, the availability of data cannot be guaranteed in traditional web architectures. Rather, one needs to trust the operators of the servers to continuously guarantee the availability of resources. In case data becomes unavailable, e.g. servers ceasing operation, the established web protocols do not support mechanisms to make data available again,

The associate editor coordinating the review of this manuscript and approving it for publication was Shajulin Benedict.

even if it were present somewhere else in the network. With resources already available over protocols such as HTTP, it remains challenging to provide the properties of *integrity* and *availability* in a decentralized setting. This limits the long-term preservation of information and the traceability of information resources. Consider here as an example the need to access particular datasets for conducting research or for enabling machines to automatically find and use data on request.

Furthermore, the exact time of the issuance of information can today not be verified for arbitrary web resources. This is, however, necessary for scenarios where the time of issuance is of primary importance, e.g. for resolving disputes on intellectual property such as patents and trademarks without having to rely on a trusted third party such as a patent office [4].

In domains where these limitations are of particular concern, specific technical solutions have emerged already. Especially in scientific research, the need for the long-term storage of information in a way that it is openly accessible and preserved in its original form has been broadly recognized in the form of the FAIR principles [5]. For example, data availability is provided through content distribution networks for websites by increasing the number and distribution of servers or by reverting to peer-to-peer approaches [6]. Software repositories are today designed as decentralized versioning systems such as Git [7] for an integrity-secured storage. In addition, centralized platforms exist, e.g. for scientific research repositories for making metadata available via Digital Object Identifiers (DOI) [8] and novel forms of publications focusing on the provision of data or software [9].

While specialized solutions might be well suited for these particular areas, the storage of data independent of a particular platform or resource location might still be desirable. Potential benefits include: less reliance on single points of failure; the ability to verify data integrity when using hyperlinks; fewer broken links; increased availability independent of central servers; and in the face of distributed-denial-of-service attacks, counteracting the gate-keeper function inherent in centralized platforms by publicly providing a resilient network of integrity-secured and available information.

A promising area addressing the limitations of availability and integrity are decentralized architectures using blockchains. Blockchains provide a single point of truth for data storage, such that integrity and immutability may be publicly verified. Attestations methods based on blockchains allow for the issuance of claims on the existence of information and their verification by a third party at a later point in time [10]. However, for eliminating single points of failure through decentralized storage, today's blockchains are not well suited. Blockchain transactions were originally designed to record the transfer of monetary amounts [11] and have been extended to carry data for smart contract calls with data types for binary data on the order of bytes to kilobytes [12], [13].

Due to the design of blockchains, consensus is typically reached on all data stored, which imposes difficulty for scalability solutions [14]. In addition, decentralized storage protocols became available for storing larger amounts of data. For example, the Inter-Planetary-File-System (IPFS) [15] provides availability and resilience by replicating data across nodes of a network. Recently, decentralized storage protocols were integrated in this approach for storing data on the web in a decentralized way [16]. Furthermore, IPFS can be combined with reward mechanisms. Such a system rewards participants for proofs of storing files using a market-based price that is derived from data size and storage duration [17].

In this paper, we thus examine the extension of an integrity-providing attestation approach while ensuring availability through multi-protocol data storage. The attestations are conducted by reverting to a blockchain, which stores the integrity metadata. For the storage of content, traditional hypertext protocols and IRIs may be used as well as novel protocols such as IPFS. As we will show, the main reason for this architecture is to decouple data from the mechanisms and metadata for verifying integrity and for guaranteeing the availability of information.

The remainder of this paper is structured as follows. Section II introduces foundations on providing data integrity, decentralized storage protocols, and blockchains. Section III describes the architecture and implementation of the approach. Section IV is an evaluation of using the approach based on performance metrics and measurements of limiting factors. Section V discusses results and potential use cases. Section VI discusses further research and concludes the paper.

II. FOUNDATIONS AND RELATED WORK

For laying the foundations for our approach, we will discuss methods for ensuring data integrity and subsequently for data distribution in web environments. Lastly, we will regard existing approaches for using blockchains to support the verification of data integrity and storage.

A. DATA INTEGRITY

Methods for providing integrity for data in transit or at rest are commonly employed in today's internet architecture [18]. At the level of information, an unchanged representation of data is required by all participants, requiring adding metadata for describing the content and its integrity [19]. This is assured through the concept of summarizing data with a function $h(d) = v_t$. When applied to data d of possibly unknown format and length, v_t is a *digest* that is computed and stored at time t, such that it might be compared to a re-computation of the function value v_{t+1} at a later point in time $t + 1$ such that $v_t = v_{t+1}$ indicates integrity. Depending on the function, there are not necessarily strong guarantees for integrity.

Further, summarization functions can be distinguished by two types: *checksums* and *hash functions*. Checksums such as cyclic-redundancy-check (CRC) are long-standing standards on the internet for efficiently detecting unintended transmission errors [20], e.g. the internet checksum of the transmission control protocol (TCP) [19]. Hash functions extend the

summarization with the goal of minimizing collisions, i.e. producing the same v for two different inputs d, d' [21]. Values must be distributed uniformly in the solution space, e.g. as used for hash tables or sorting algorithms.

Providing strong integrity guarantees requires further properties offered by cryptographic hash functions such as SHA-2, SHA-3, or Keccak-256 [22]–[24]. Integrity is thereby achieved through three fundamental properties. It should not be feasible to (1) find d given v (*pre-image resistance*), (2) find for any given input d another input d' resulting in the same v (*second pre-image resistance*), and (3) finding any two inputs d and d' resulting in the same v (*collision resistance*).

Given these properties, hash functions are used to verify the integrity of data for detecting technical errors, as well as in untrusted environments where data is at rest or transferred over an insecure channel. In practice, the relevance of cryptographic hash functions is evident on a technical level, e.g. for securing commits in version control systems [25], the verifiable replication of data in blockchains [13], and on the level of information where research repositories such as Zenodo [26] use SHA-2 values together with further metadata for realizing the FAIR principles of findability, accessibility, interoperability, and reusability [5].

B. DATA DISTRIBUTION

Data distribution on the web can be realized using centralized and decentralized protocols. In the following we regard in particular the properties of integrity and availability of these protocols.

1) CENTRALIZED PROTOCOLS

Traditional web architectures are built on the TCP/IP stack, using application-level protocols such as the file transfer protocol (FTP) or the hypertext transfer protocol (HTTP), including its recent successors HTTP/2 [27] and HTTP/3 [28].

The protocol suite is based on the client-server model where a client sends an HTTP request to a server-side resource location, identified through a URI[1] [29] or an IRI [30]. An according HTTP response is then sent to the client. This principle model remains unchanged also with the recent innovations in HTTP/3 and QUIC which optimize performance and load balancing, e.g. through the dynamic selection of server locations with minimal latency [31].

For combining these protocols with mechanisms for securing integrity, few solutions exist and rely on the additional transmission of integrity metadata. One example is the concept of Trusty URIs [32], a proposal to include verifiable hash values in links. In this approach, a link to a resource additionally contains integrity metadata in the form of a checksum value, which is verified by the client. This approach illustrates the need for storing resources separately from their integrity metadata. However, the integrity guarantees so far do not extend beyond the correctness of the transmission since data and integrity metadata are transmitted through the same channels. This limits the use of checksums and hash functions in untrusted environments with unknown participants.

In summary, we can state that within the class of centralized protocols, there exist single points of failure, introduced by addressing resources through individual identifiers. Furthermore, availability is limited due to individual servers. Integrity guarantees are not permanent and cover only individual transmissions between clients and servers.

2) DECENTRALIZED PROTOCOLS

Decentralized protocols are either based on *client-server-* or *peer-to-peer-architectures* [19], [33]. Thereby, a distributed protocol execution enables the storage of data at the nodes of a distributed system. We will illustrate the main properties of these architectures in the following by reverting to concrete examples.

A well-known example of a client-server-based architecture is the distributed version control system (DVCS) Git. In the case of Git, the state of a data repository on the client-side is captured via operations such as *commit* [7]. Versions and sequences of versions derived in branches determine the state which is transferred with the *push* operation to any number of remote locations addressed by a URL through HTTP or other protocols. The inverse operations *fetch* and *merge* retrieve and join data in a local repository. Even though Git is client-server based the system state is synchronized throughout the distributed system.

The Git architecture requires data synchronization between individual peers or between a centralized server and multiple peers. Metadata is provided in the form of version identifiers, branches, tags, and hash values. Hash values are used for the identification of information, attribution to authors, and notably, the verification of integrity. Data exchanges of a client are limited to the distinct endpoints of the used repository. Furthermore, *commit* and *push* are explicit operations performed by the client for regularly capturing the state of data and software repositories. The retrieval of individual and previously unknown resources is not directly supported. Rather, it is necessary to fetch or clone individual repositories known by a URL beforehand.

While it is possible with Git to add multiple remote repository branches, availability is not inherent in the architectural design. Each participant maintains a list of known repositories individually, merges the local state of a repository separately, and resolves conflicts between multiple remote states and the local state. Instead of algorithmic consensus, consensus is an emergent phenomenon established manually when needed. As the architecture is targeted at software development, it requires individual data synchronization between peers or, more commonly, between a centralized server and multiple peers.

Protocols on the basis of peer-to-peer networks eliminate the differentiation between client and server roles. Peer-to-peer protocols such as BitTorrent [34], [35], Swarm [36], or the Inter-Planetary-File-System (IPFS) address data in a content-based fashion, independent of the location in the

[1] https://www.w3.org/TR/uri-clarification/#classical

network [15]. For retrieving files from the network, a unique ID based on hash values is sent to connected nodes in the network from where it is distributed through gossiping [37]. Any node in the possession of individual parts of a file can answer the request and serve them to the requesting node, which subsequently creates another replication.

The BitTorrent approach is centered around the individual retrieval of specific sets of files or directories known as torrents [35]. In the fashion of a torrent, the retrieval occurs in parallel from any number of nodes. This leads to higher utilization of bandwidth and higher availability when compared to client-server approaches such as HTTP or Git. For the retrieval, the corresponding hash-based identifier is required and needs to be obtained individually and explicitly. Therefore, the retrieval is non-interactive and does not support references or links from one torrent to another.

IPFS possesses, similarly to BitTorrent, a peer-to-peer architecture allowing for the retrieval of specific sets of files or directories [15], [16]. In addition, it provides mechanisms of a file system within this architecture, including references and the fine-grained retrieval of blocks of files. In IPFS, files or directories are stored as objects, addressed with a content identifier (CID). A CID is computed based on the content of objects and can act as a link. It is created in a standardized format or *codec* that describes the content-based identification method, such as the hash function, the hash value of the objects, and the data format [38].

An IPFS object may be retrieved from a network participant and stored locally as a node of a directed acyclic graph (DAG). The DAG links files using their CID in an integrity-secured fashion using a Merkle tree hashing method. Non-hierarchical structures may be formed through links for realizing hypermedia structures. This can be accomplished, for example, by loading an HTML file that subsequently loads further site contents through CID references.

IPFS objects are cached temporarily upon access, such that they become available to other nodes. In addition, they are cached offline for local use. For achieving availability, objects need to be distributed to a sufficient number of nodes where they can be 'pinned' for permanent storage. This content-based addressing and linking through content identifiers of objects in a DAG lead to higher resiliency, availability, and integrity. IPFS enables interactive browsing on distributed objects under the condition that they are available from at least one node in the network.

Such an architecture allows, for example, to realize decentralized research repositories, where publications are stored by individual researchers, institutions, and others without a dedicated central infrastructure [39].

In summary, the class of current decentralized protocols offers, on the one hand, client-server protocols, which enable data synchronization between individually participating nodes such as in Git. In this case, data exchanges are initiated separately and through individual requests and responses between pairs of nodes. Such protocols provide integrity and limit the consistent distribution of information across the whole network. Availability depends on the initiation and maintenance of data exchanges.

On the other hand, decentralized protocols based on the peer-to-peer paradigm distribute the control over storing and retrieving files across the network. These types of protocols provide integrity and higher degrees of availability since the exchange of data does not depend on the manual coordination of nodes. Single points of failure are still possible in this architecture as availability is controlled by individual nodes. This means that availability rests on the explicit choice of persistent storage over transient storage, e.g. through caching settings and pinning in IPFS.

C. BLOCKCHAIN-BASED PROTOCOLS

Blockchains are data structures used in combination with consensus protocols for the distributed storage of immutable data transactions due to their verifiable inter-linkage in blocks and the verifiable protocol execution [40]. Any participant in such a distributed network is able to verify the data integrity of the chain. Individual transactions are signed using private keys, sent with corresponding public keys for identifying the sender, and addressed with the public key of one or more receivers.

Smart contracts extend this concept to the execution of programs stored within a block and executed together with the consensus logic of the underlying protocol [41]. This system design permits the operation of permissionless blockchains, which are openly accessible and governed by algorithmic consensus. A ledger is created that acts as a single point of truth. This is in contrast to storage protocols such as IPFS, which maintain any number of arbitrarily different files or directories. Therefore, the operation of such systems does not depend on the control of individual nodes. However, they are limited in scalability primarily by storage capacity, throughput, cost, and latency [14]. Blockchains are well-suited for tracking the storage of data in transactions that do not depend on trusted third parties or intermediaries, where the storage of data might be located within or outside of the blockchain. For financial transactions and comparatively small amounts of data in smart contracts, the permissionless systems Bitcoin and Ethereum today store data on the blockchain itself [13], [41].

Within this paper we focus on the Ethereum blockchain for its smart contract capabilities. From a client perspective, the creation of an Ethereum smart contract involves its development in a high-level programming language, the compilation to byte code, and the transmission through a transaction to the blockchain [40]. The Ethereum protocol autonomously creates a *contract account* with a specific address where function calls might be sent in subsequent transactions. In contrast to a so-called *externally-owned* account of a participant, there exist no public and private keys for a contract account, limiting its ability of sending transactions to the programmed functions. The main characteristic of *contract accounts* is, therefore, the autonomous execution of smart contract programs. The state of individual executions is persisted in

each block following a function call, such that participating nodes can locally execute all smart contract calls in order to verify their correct operation. Therefore, smart contracts permit the storage of system-external data and can be used for implementing so-called *attestations* where evidence of the existence of documents or data is recorded [42].

Attestations based on smart contracts persist the hash value of data such that (1) the *prior existence* of the data, (2) the address of the *account owner* of the transaction, and (3) an *approximate timestamp* can be verified at a later point in time by anyone with access to the openly available blockchain data [10]. Blockchain-based attestations might be used to publicly prove the existence of information, documents, or intellectual property. While the storage of data is limited to relatively small sizes not permitting general file storage distribution, data stored on blockchain-external systems might be the subject of an attestation.

The combination of blockchain protocols with facilities for storing larger amounts of data has only been treated recently. In the work by Liang *et al.* the concept of ProvChain is described, which permits the auditing of all data access in a cloud storage application by storing provenance and operation data on a blockchain [43]. Li *et al.* proposed the concept of IntegrityChain as a decentralized storage framework that permits to prove data possession using a blockchain [44]. For this purpose, they set up their own blockchain platform and described it formally. Bian *et al.* revert to permissioned blockchains based on Hyperledger Fabric for realizing a decentralized patent application system [4]. As the data involving patent applications can become very large, they revert to IPFS as external data storage. Recently, approaches have been proposed for storing semantic data using blockchains. Apart from the attestation of ontologies by using blockchains, which involves only little storage space [45], the ColChain approach has been proposed to store smaller fragments of RDF data on a blockchain-like decentralized architecture [46]. The latter approach employs a user-based consensus protocol that is not automated. In the approach of Knowledge Blockchains, also a specific consensus protocol has been proposed that can, however, be automated. This approach has so far been described for storing enterprise models and ontologies on-chain but has limitations in terms of storage scalability [47], [48].

D. DISCUSSION OF CURRENT LIMITATIONS

In their current state, the concept of attestations is limited to the recording of evidence, independent of the transfer or storage of the underlying data. The attestation has to be performed manually and explicitly at the sender or operator of the storage facility. When retrieving the data, the attestation requires manual validation of the evidence, performed for every data transfer or storage retrieval. Through this procedure, the three properties of existence, owner, and time are verified in order to ensure that the original source provides data or that it is retrieved in the way originally intended. Therefore, the limitations of attesting stored or transferred data with current methods are:

1) The incompatibility with existing data storage and transfer protocols such as HTTP, Git, and IPFS limits distribution and availability.
2) The manual and explicit creation of attestations per transfer or storage invocation at the source and, conversely, the manual and explicit validation at every receiver.
3) The source-based validation, which is bound to the originator as the only source and not to other parties possibly also in the possession of the data.

In summary, state-of-the-art web protocols as well as current attestation approaches based on blockchains are only partially providing data integrity, distribution, and availability. While blockchains by themselves would support these properties in combination, their scalability is insufficient for data storage. Currently, neither web protocols nor blockchains support an integration where integrity is secured by blockchain-based attestations with the distribution and availability of current (de)centralized web protocols.

III. ATTESTED MULTI-PROTOCOL LINKS

In light of the limitations for providing data integrity, distribution and availability with existing protocols such as HTTP, Git, and IPFS, we propose a concept that combines multiple existing protocols with blockchain-based attestations. Based on a previously introduced attestation concept [10], the following architecture offers an integrated approach by a. performing blockchain-based attestations for securing data integrity, b. by introducing a format for creating and validating links automatically, and c. by allowing for validation and re-distribution with multiple protocols independent of the source in order to allow for a higher degree of distribution and availability independent of the source.

The attestation concept builds on existing ideas for a more decentralized, trustworthy, and available web. In this area, the ''web of trust'' [49] has been discussed as well as aforementioned protocols for decentralized storage [50] and ideas of a permanent web [15]. The aim of providing technical properties through an attestation approach therefore aims at establishing decentralized, trustworthy, and available systems for distributing information.

In the following sections, the core concept with its aims and the underlying technology are introduced first, followed by the architecture and its implementation.

A. MULTI-PROTOCOL STORAGE AND ATTESTATION CONCEPT

Attestations in general aim at establishing trust by presenting evidence for the existence of information to another party or any number of untrusted third parties [42] and through using blockchains in the case of blockchain-based attestation [51]. Specifically, within the suggested architecture, attestation is understood as (a.) the creation of a claim about the existence

of information, bound to an identity, and (b.) the validation of the claim by any other identity at a later point in time.

The distribution of information pursues decentralization and availability. Due to the use of blockchains, the system architecture is a distributed system [19], in particular, a decentralized system applying non-centralized control over storing information in a replicated manner for availability.

Technically, an architecture realizing blockchain-based attestation and distribution should address the outlined limitations (Section II-D) of existing web protocols and attestations for providing data integrity, distribution, and availability in the form of the following requirements:

- *Providing data integrity with existing web protocols.* Using existing web protocols, the integrity-secured transfer or storage of data must be possible. Due to the widespread use of protocols such as HTTP and Git and the vast amount of information already stored on the web, the preservation of integrity must be supported by existing protocols without requiring the adoption of new protocols. In particular, information must remain unchanged in relation to its original publication.
- *Validation through a trusted global state instead of source-based validation.* Instead of performing validations of attestations per transfer or storage invocation based on the source, the validation must be performed against a trusted global state, i.e. a state representation that is transparently and openly visible for all parties involved. Without consultation of the source, a claim on the existence of information, the approximate) publication time, and a source identifier must be verifiable.
- *Creating and validating attestations automatically at the level of links.* The unchanged re-distribution of data must be possible after publication for availability. For this reason, the validation of an attestation must be independent of the original location of the data and occur automatically. Given existing web protocols, the only identifiers independent of the actual data are links. It must be possible to establish the validity of an attestation when given one or more links.

With an attestation and distribution based on links, the architecture requires an additional layer above web protocols for attestation through links.

B. MULTI-PROTOCOL STORAGE AND ATTESTATION ARCHITECTURE

Based on the stated high-level requirements, a possible system architecture can be derived from the following architecture requirements:

- *Multi-protocol support with content-based identifiers.* The requirement of providing integrity with existing web protocols can be satisfied by creating identifiers based on data content, regardless of the protocols and address of the data. Content-based identifiers are abstract from addressing and require assignment to protocols and addresses compatible with today's web, e.g. in a standardized form through URI [29]. For example, HTTP and Git might be used in a content-addressable fashion in addition to protocols supporting this concept natively, such as IPFS [15].
- *Blockchain-based attestation for providing a trusted global state.* One possibility is the use of blockchains for representing a state that is transparently and openly visible for all parties involved. Validating attestations against this state ensures independence from the source. Technically, the recording of claims can be performed with an attestation smart contract [51]. The state of the contract transparently holds for each claim at least one record containing its content-based identifier, URIs with further storage metadata, the blockchain identity of the issuer, and a timestamp.
- *URI scheme and link format of the structure of a link pointing to a claim.* For attestations and validation at the link level, a format allowing for the standardized processing of links is required. The link format structure must include a content-based identifier and might optionally include a semantic identifier. Semantic identifiers are necessary for persistent links to content changing over time. Client-side validation can occur automatically when following a link [32] by retrieving the data and validating the attestation through cryptographic hash functions and Merkle trees [11], [52].

Figure 1 gives an overview of the main components of the architecture. The following sections describe (1) the general system structure and the process of storage and attestation. Subsequently, the details on the process are presented, including (2) the distribution of files, (3) the issuance and recording of claims, (4) the creation and resolution of links, and (5) the retrieval and validation of claims. An attestation is present on the blockchain upon completion of (2) and (3). Links might optionally be present in the system after (4). The final validation of an attestation is the consequence of (5) and produces a valid or invalid attestation result.

1) SYSTEM STRUCTURE

For achieving modularity, the system has been designed in the form of several components for carrying out attestations. First, a *Claim Issuer* (CI) initiates the distribution of files and the issuance of claims. Then, a *Claim Validator* (CV) retrieves files and validates corresponding attestations. No trusted relationship between CI and CV is assumed. Furthermore, CI, CV, or any untrusted third party are assumed to engage in the creation of links.

The following assumptions are made for the components of the architecture: Corresponding to the architecture requirement of supporting multiple protocols, (1) a storage network is assumed where an arbitrary number of nodes provide services through web protocols, encompassing state-of-the-art centralized and decentralized protocols (Section II-B). Nodes are assumed to be distributed, independent from each other, and reachable over private or public wide area

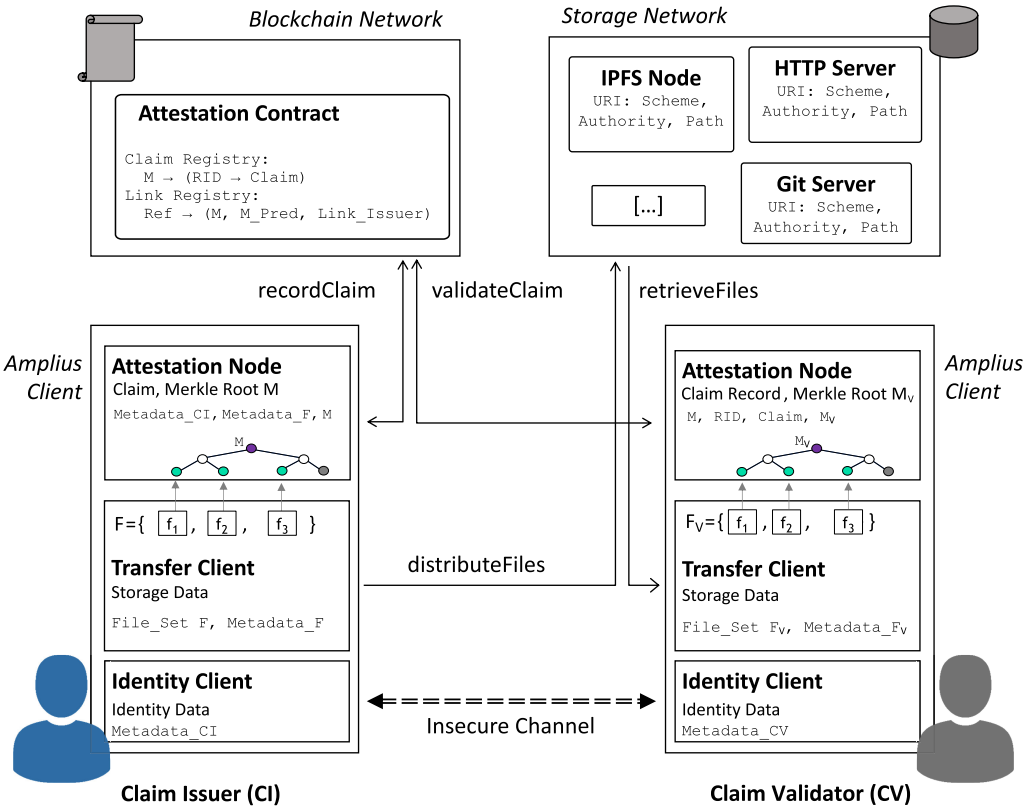

FIGURE 1. Decentralized attestation and storage application architecture.

networks. Related to the attestation, (2) a blockchain platform supporting smart contracts is assumed where an attestation smart contract can be deployed, e.g. on Ethereum. Due to the properties of blockchains, the autonomous execution of the smart contract is outside the control of CI, CV, and third parties. The functions of the smart contract will be discussed in detail in the following sections. (3) The URI scheme and link format and a client-side implementation required for automated validation are finally assumed. For validation purposes, a prototype implementation of the client realized the attested multi-protocol link immutability system (amplius).[2] It supports the functions outlined in the following sections. The coordination of an attestation is managed in a decentralized fashion by the client in combination with the smart contract.

2) DISTRIBUTION OF FILES IN THE STORAGE NETWORK

Initially, a file set F is locally available at the claim issuer CI. F contains any number of files for distribution through the storage network of a set of individual nodes N. For distribution and attestation on a per-file basis, F might contain only one element.

Each client-initiated transfer operation involving files $F_v \subseteq F$ and node $n \in N$ is represented by the function

$$distribute : F_v, n \rightarrow (Scheme, Authority, Path)$$

which yields a tuple representing a single URI. The set of all URI tuples is denoted as URI_Set. No restrictions are imposed besides URI-based addressing. However, protocols might impose additional restrictions such as addressing F under a single repository URI, e.g. when using Git. There is no requirement for content-based storage on the protocol level.

3) ISSUANCE AND RECORDING OF CLAIMS

An attestation consists of a claim issued by claim issuer CI and the validation performed by claim validator CV. Thereby, a claim documents (1) the existence of F, (2) the possession of f by CI, and (3) the time of recording the claim $Timestamp$.

a: HASH-BASED COMPUTATION OF CONTENT-BASED IDENTIFIERS

The requirement of identifying files by their content is realized through a cryptographic hash function such as $SHA3$[3] applied in a Merkle tree hashing algorithm. In this way, the Merkle Root M computed through $MerkleTree(F) = M$ acts as a content-based identifier of F such that it can be calculated without additional information.

In particular, the algorithm initially partitions F in pairs of files $\{f_1, f_2\} \subseteq F$ in order to calculate the hash function H of the concatenation $H(f_1||f_2) = v_1$. In the same fashion, the

[2] A preliminary version is available at https://github.com/fhaer/amplius

[3] See https://doi.org/10.6028/NIST.FIPS.202

resulting values are repeatedly partitioned into pairs $\{v_1, v_2\}$, concatenated and hashed with $H(v_1||v_2) = v_{1||2}$ until there exists one resulting hash value denoted as Merkle root M. Before each partitioning step, the case of an odd number of files or values is handled by adding an additional element $v_0 = 0$. Furthermore, this hashing method assumes defining a total order over F through an index set that can be implicitly generated by sorting files by their content-based identifiers.

b: GENERATION OF KEY PAIR AND ACCOUNT ADDRESS

Interactions with the blockchain require for CI the computation of a blockchain account address. Due to the nature of permissionless blockchains, the generation of an externally-owned account identified by $Address_{CI}$ of the claim issuer is carried out locally without network interaction. The address is derived from a randomly generated public-private key pair. Similar to externally owned accounts, the private key is persistent data stored locally by CI.

c: METADATA AND ENCODING

Corresponding metadata such as author identities or file attributes might be present. The separate processing of metadata for the possibility of building repositories entails the creation of separate identity metadata and storage metadata of the file set in two tuples $Metadata_{CI}$ and $Metadata_F$ respectively.

Identity metadata is comprised of public data suitable for the pseudonymous identification of CI. Therefore, the tuple $Metadata_{CI} = (Address_{CI}, DID_{CI})$ contains the address as an identifier in interactions with the smart contract in addition to DID_{CI} as a decentralized identifier [53]. A DID is a standardized identifier in a specific format suitable for the global identification of network participants. DID are used predominantly in decentralized networks and allow participants to retain and self-manage real-world and pseudonymous identities. The projected use of these identities spans across multiple attestations in addition to other networks, websites, or services.

Storage metadata of a file set F includes the MIME types $MIME$ [54], a timestamp and the URIs of the storage network represented by the elements of $Metadata_F = (MIME_F, Timestamp_F, URI\_Set_F)$. Given this information, a repository of file sets might be created for the general case. Regarding the URI format, each URI is stored as a tuple $(Scheme_F, Authority_F, Path_F) \in URI\_Set_F$ in order to distinguish the protocol and other parts of the URI. Conceptually, the elements are one or more addresses belonging together for the purposes of distribution, e.g. a set of image files.

The tuple is adapted for specific domains requiring additional data, e.g. for including Digital Object Identifiers (DOI) [8] in scientific publications.

The elements of both sets will be stored in the smart contract and therefore require the application of an appropriate encoding supported by the blockchain network. In the case of Ethereum, the Ethereum Virtual Machine (EVM) can execute the smart contract with $Metadata_{CI}$ and $Metadata_F$ encoded in arrays of 32 bytes. Here, a character encoding might be applied, followed by splitting the result into sequences of 32 bytes.

d: RECORDING OF CLAIMS

The recording of claims is conducted by a smart contract identified through a constant and well-known address on the blockchain. With the local creation of a Merkle root M, $Metadata_F$ and $Metadata_{CI}$, CI invokes the smart contract function *recordClaim*. The function stores $Timestamp_F$ of the current block in $Metadata_F$, assigns for $(Metadata_F, Metadata_{CI})$ a record identifier RID and assigns RID to the Merkle Root M. As a result, an individual claim record $Claim = (Metadata_F, Metadata_{CI})$ is stored and retrieved by a smart contract function globally in the form:

$$claim\_record : M \rightarrow (RID \rightarrow Claim).$$

Additional availability of F can be provided by issuing additional claim records for M involving another URI_Set'. In the spirit of decentralization, the subsequent recording of claim records for M is not limited to CI due to the ability to validate the integrity of F against M. The smart contract is able retrieve the corresponding claim records given M; therefore, M is the basis of links.

From the point-of-view of CI, the issuance and recording is concluded after RID is retrieved by a locally available and fully validating blockchain node that returns RID from a block having at least n successors. Depending on the blockchain, the security parameter $n \geq 0$ prevents double-spending attacks [55].

4) CREATION AND RESOLUTION OF LINKS

Links are references to claims and therefore provide access to the corresponding files in the storage network.

a: RECORDING OF LINKS IN THE SMART CONTRACT

In the fashion of domain names of the world wide web, links point to content changing over time using semantic references such as names or also globally unique IDs in the form of UUID.[4] For this reason, a link is a reference Ref to Merkle root M that can be updated over time by the issuer of the link. In the smart contract, a link is created and stored through

$$Link : Ref \rightarrow (M, M_{pred}, Address_{LI})$$

with link issuer address $Address_{LI}$ and M_{pred} as the predecessor of M such that a chain of claims is created over time, preserving access to prior versions.

b: RESOLUTION OF LINKS

Any Ref registered with the smart contract yields all records of a claim corresponding to M. In order to retrieve the specific claims issued by an $Address_{LI}$, a filter selecting claims where $Address_{CI} = Address_{LI}$ is applied. In this way, the claim that is the original source of a link is established.

[4] See https://tools.ietf.org/html/rfc4122

c: URI SCHEME AND FORMAT

Interoperability with existing protocols and networks such as the world wide web requires the definition of a URI scheme and format for claims and links. For clients retrieving files and validating attestations automatically, three methods are assumed. A client might be in possession of (1) a Merkle root M from the issuance of a claim. (2) It might possess a link *Ref* created in the format of a global identifier, specifically in *UUID* format. (3) It might possess a link created with a semantic identifier created as a globally unique *Name*. The choice of the format is up to the link issuer.

(1) `amplius://<Contract_Address>/M/<M>`
(2) `amplius://<Contract_Address>/U/<UUID>`
(3) `amplius://<Contract_Address>/N/<Name>`

The client-side implementation with the smart contract deployed at *Contract_Address* is invoked by the scheme using the given format.

5) RETRIEVAL AND VALIDATION OF CLAIMS

The final part of the attestation concerns the retrieval of claims. Based on the content retrieved through a claim from the storage network, the validity of the attestation result is determined.

a: RETRIEVAL

Claim records are globally available through the attestation smart contract. Given the Merkle Root M of a file set, the corresponding claim records are obtained according to the mapping discussed in Section 3d). Based on the stored record IDs, $Metadata_F$ and identity metadata $Metadata_{CI}$ provide the basis for the content-based validation. From the storage network, a file set F_v is retrieved through the URI_Set_F of any $Metadata_F$. The validation can be conducted with at least one $Metadata_F$ by a client if availability is given, i.e. $\forall u \in URL\_Set_F.Available(u)$.

There could be, for example, a file set of two image files with MIME type PNG, timestamp 2021-01-31 at 14:42:43 retrievable through HTTP at https://example.ch/i1.png and https://example.ch/i2.png. In this case, the storage metadata is defined (see Section 3c) by

$$
\begin{aligned}
&Metadata_F \\
&= (MIME_F, Timestamp_F, URI\_Set_F) \\
&= (\texttt{'image/png','2021-01-31T14:42:43',} \\
&\quad \texttt{\{('https', 'example.ch', '/i1.png'),} \\
&\quad \texttt{('https', 'example.ch', '/i2.png')\})}
\end{aligned}
$$

Note that elements of the URI_Set_F are tuples distinguishing the protocol from other URI parts (see Section 3c). In an implementation such as the amplius prototype, compression should be applied for preventing the redundant storage of the URI parts.

b: VALIDATION OF CLAIMS

For validating a claim given M, the hash-based computation of this Merkle root is first computed using F_v. Based on this result, the validation with the attestation is conducted by the smart contract and the client of CV.

With F_v, the Merkle root is computed on the basis of a hashing algorithm. Thereby, the elements of F_v are individually applied to a hashing function, i.e. $H(f_v) = v$ is computed for each file $f_v \in F_v$. The resulting values are leaf nodes of the Merkle tree. With it, the algorithm outlined in Section 3a) is applied by repeatedly hashing concatenated pairs of hash values until a single value Merkle root M_v is reached. The algorithm accounts for the case $|F_v| = 1$ by constructing the hash value based on $H(f_v) = M_v$ without the explicit construction of a tree. This case arises if a single file is distributed where the file set contains one element.

The result of the validation depends on the F_v retrieved through the storage network. In case $M = M_v$, integrity of F_v is assumed, leading to a valid attestation result. Else, the attestation result is invalid. The validity of the attestation implies the presence of the original file set $F_v = F$, establishing the prior existence of F.

Suppose, for example, a claim is recorded for a set of image files under Merkle root $M = c9e8a[\ldots]77f15$, a SHA-256 hash value with a length of 256 bit, here abbreviated and in hexadecimal notation. The claim record is stored and retrieved by a smart contract function taking M as a parameter (see Section 3d). For validation with a given M, the record ID *RID* and *Claim* are obtained through this function. First, the file set F_v is retrieved from the network using the storage metadata $Metadata_F$ of the tuple *Claim* (according to the previous subsection). Secondly, the validation is carried out. Suppose files i1.png and i2.png have been retrieved as elements of the file set F_v. In this case, M_v is computed from the Merkle tree with the two files as leaf nodes, here consisting of $M_v = H(H('i1.png')||H('i2.png'))$ with SHA-256 as hash function H. For example, with abbreviated hash values, the computations required are $H('i1.png') = 1a4c7[\ldots]f1499$ and $H('i2.png') = a4bda[\ldots]7c056$, and finally $M_v = H(1a4c7[\ldots]f1499a4bda[\ldots]7c056) = c9e8a[\ldots]77f15$. The computation is recursive in the case of more files (see beginning of current Section). Formally, the attestation result is valid here since $M_v = M$ could be established.

In case the attestation result is valid, further information can be obtained from the claim record metadata. In particular, the timestamp of the claim record is obtained from $Metadata_F$ and the pseudonymous identity of the claim issuer *CI* from $Metadata_{CI}$. Given the identity metadata, the blockchain address and *DID* can be retrieved to link the identity across multiple attestations or other *DID* occurrences if the issuer chooses to manage it, e.g. in other attestations or in user accounts on websites (see Section 3c). The client, therefore, obtains the file set $F = F_v$, the attestation of the prior existence of the file set, the issuer of the claim with address and identity information, and the timestamp showing the date and time of the recording of the claim.

C. PROTOTYPE IMPLEMENTATION

For demonstrating the feasibility of the concept and for evaluation, the system has been implemented as a prototype.

Figure 2 shows the software architecture of the application from the client point of view. On the application layer, a client has been implemented in Python 3.8, with modules for the Identity Component, Attestation Component and, Transfer Component. The client uses the Web3 API for interfacing with the Ethereum blockchain on the network and consensus layer. There, an attestation smart contract written in Solidity is deployed and used with externally owned accounts. Files are stored on HTTP, Git, and IPFS servers rented for this purpose. The UML sequence diagram in Figure 3 summarizes the distribution, issuance and link creation with the prototype. Figure 4 summarizes the retrieval and claim validation. The individual components of the prototype are discussed in the following paragraphs.

The transfer component takes as input files or other data objects that are used for distribution with the protocols HTTP, Git, and IPFS. Conversely, files can be retrieved through these protocols with the implementation. The client is shown, including these functions, in Figure 5 with a command line interface.

For the attestation of files after distribution, the attestation component carries out Merkle tree computations with the SHA-256 hash function based on the files and encodes the result with the *URI_Set* obtained from the transfer. The encoding includes compression and data format changes for the issuance and validation of claims through the attestation smart contract. Up until this point, computations occur locally within the client software.

As part of the network and consensus layer of the prototype, the attestation smart contract has been written in the Solidity programming language for the Ethereum blockchain. Source Code Listing 1 shows the main data structures for recording claims according to Section III. In conjunction with the client, the interaction is discussed in the following paragraphs.

For interfacing with the blockchain, the identity component generates externally owned accounts for Ethereum. Any user interfacing with the software initiates the generation of an account with the client. As shown in the UML sequence diagram in Figure 3, the implementation uses the Web3 API in order to generate accounts locally through a random generation of a private key, the derivation of a corresponding public key, and the derivation of a blockchain address. As a result, the public-private key pair is used for sending attestation transactions to the smart contract. Thereby, the claim issuer *CI* signs each transaction with the private key. Once a transaction is confirmed, the address is the publicly visible identity of *CI*, registered for the claim issuer with the smart contract. Analogously, a transaction is signed and sent when a link is newly created and stored with the smart contract. For resolving a link, read-only smart contract functions are called, which do not modify the blockchain and do not require signatures or the creation of transactions. This is also the case for the validation of claims such that permissionless link resolutions and validations are possible. Since the blockchain is only read, the execution of these functions is instant in contrast to transactions requiring the inclusion in a block.

Source Code Listing 1: Smart contract implementation of claim and link assignments and tuples in mappings and structs of the Solidity language

```
1  contract Amplius {
2      mapping (bytes32 => Claim) public claimRegistry;
3      mapping (bytes32 => Link) public linkRegistry;
4      mapping (uint8 => bytes32) public mimeRegistry;
5      mapping (uint8 => bytes32) public
         schemeRegistry;
6      mapping (uint16 => bytes32) public authRegistry;
7      uint8 public nMimeTypeRegistry = 0;
8      uint8 public nUriSchemeRegistry = 0;
9      uint16 public nUriAuthorityRegistry = 0;
10     struct Claim {
11         mapping (uint8 => IssuerMetadata)
             issuerMetadata;
12         mapping (uint8 => StorageMetadata)
             storageMetadata;
13         uint8 nClaimRecords;
14     }
15     struct Link {
16         bytes32 merkleRoot;
17         address linkIssuer;
18         bytes32 predMerkleRoot;
19     }
20 ...
```

Regarding the attestation component and its interaction with the Ethereum blockchain, the component uses the Web3 API for calling the smart contract. As shown in the UML sequence diagram in Figure 3, the recording of a claim takes place by sending the Merkle root *M* and the *URI_Set* for retrieving a record ID (RID).

Storage metadata is divided in separate entries within mappings and structs, shown in Source Code Listing 2. The listings show only the beginning of 278 lines of the contract, consisting of 5 mappings, 5 structs and 18 functions implementing the specification functionally complete. Design choices to this end as well as limitations are discussed in the following. The contract is deployed and can be publicly called through the Ethereum address 0×5627da24A01B5799AbC84300ACBf2A778933bEed.[5]

Upon issuing an attestation, the creation of links pointing to it becomes possible. This function is also realized through the attestation component, as shown in the UML sequence diagram in Figure 3. The link will be created by the user specifying a link ID in UUID format, as a randomly generated globally unique ID using UUID version 4, or in the form of a semantic name, required to be globally unique. The assignment of semantic names might be used similarly to

[5]C.f. https://etherscan.io/address/<address>

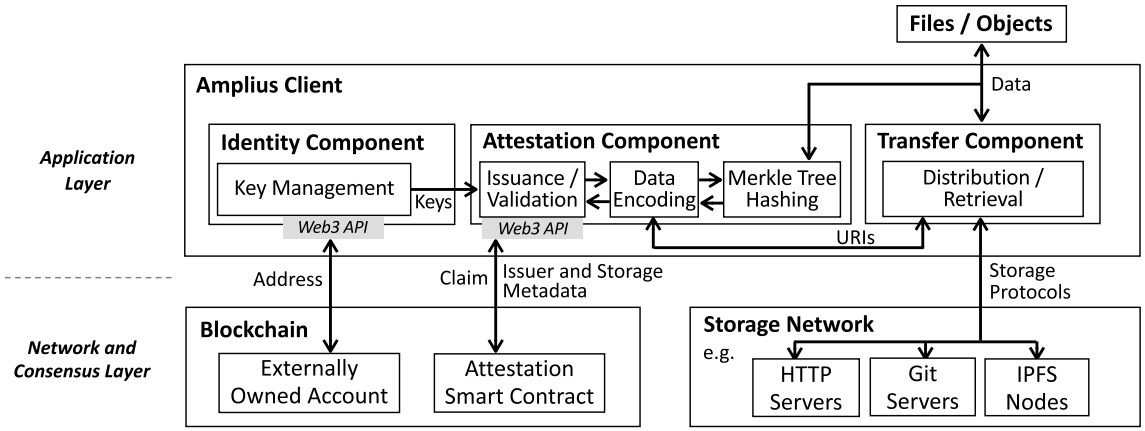

FIGURE 2. Attestation and distribution architecture for each client connecting to the blockchain and storage network.

Source Code Listing 2: Smart contract implementation of the specified metadata tuples in structs using the Solidity language

```
20 ...
21     struct StorageMetadata {
22         uint16 timestamp;
23         uint8 mimeTypeID;
24         mapping (uint8 => URI) uri_set;
25     }
26     struct URI {
27         uint8 schemeID;
28         uint16 authorityID;
29         bytes32 path;
30         uint8 fileSetExtensionMarker;
31     }
32     struct IssuerMetadata {
33         address accountAddress;
34         bytes32 did;
35     }
36 ...
```

domain name system records. According to the specification, the addition of new attestations under the same UUID or name is possible with the preservation of prior attestations retrievable through Merkle root M.

For the interaction with the Ethereum blockchain through the identity and attestation components, the Web3 API connects to an Ethereum node. Due to the attestation approach with the possibility of validating attestations at the client-side, no assumptions can be made by the client with regard to the trustworthiness of remote blockchain nodes. For deriving meaningful attestation results, running an Ethereum node locally in a fully-validation configuration is therefore required.

An integration of the client software with web browser applications is realized through the registration of the URI scheme outlined in Section III-B4 with the local operating system. Following such a link in a browser initiates the retrieval of files from the storage network, the validation of the claim, and the display of the file content in the case of validity. The process is visible in the UML sequence diagram, shown in Figure 4.

As a result, the prototype implements the functions defined by the specification. In practical terms, several implementation choices were made for demonstrating functionality with high efficiency, however, with limitations present regarding security. In particular, the functions defined by the specification, such as record claim, are split into several functions and possess additional functions for compression and format conversions. This concerns conversions of hash values between different representations of SHA-256 stored in bytes32 arrays, and the compression of URIs such that redundant parts of consecutive elements of a URI set are not stored redundantly. Sets and tuples are stored according to the specification, however, using mappings for optimizing retrieval times and gas costs in Ethereum. The security of the smart contract and client are not fully tested since the aim of this prototype is primarily an evaluation of feasibility. Limitations regarding efficiency, such as transactions costs, will be discussed separately as part of the evaluation in the next section.

IV. EVALUATION

For evaluating the attestation approach, we revert in the following to a use case in the domain of education. The goal of this evaluation is to assess the performance of the approach in terms of time and transaction costs for carrying out attestation and link operations. For this purpose, we conducted measurements using the Ethereum mainnet and the prototype implementation.

A. EVALUATION USE CASE

Today, the certification for the completion of higher-education courses, study programs or industry certifications is relevant not only offline but predominantly on the web. Commonly, qualifications and skills are referenced on career websites such as LinkedIn or for individual job applications on job portals and in individual e-mails. In these cases, the

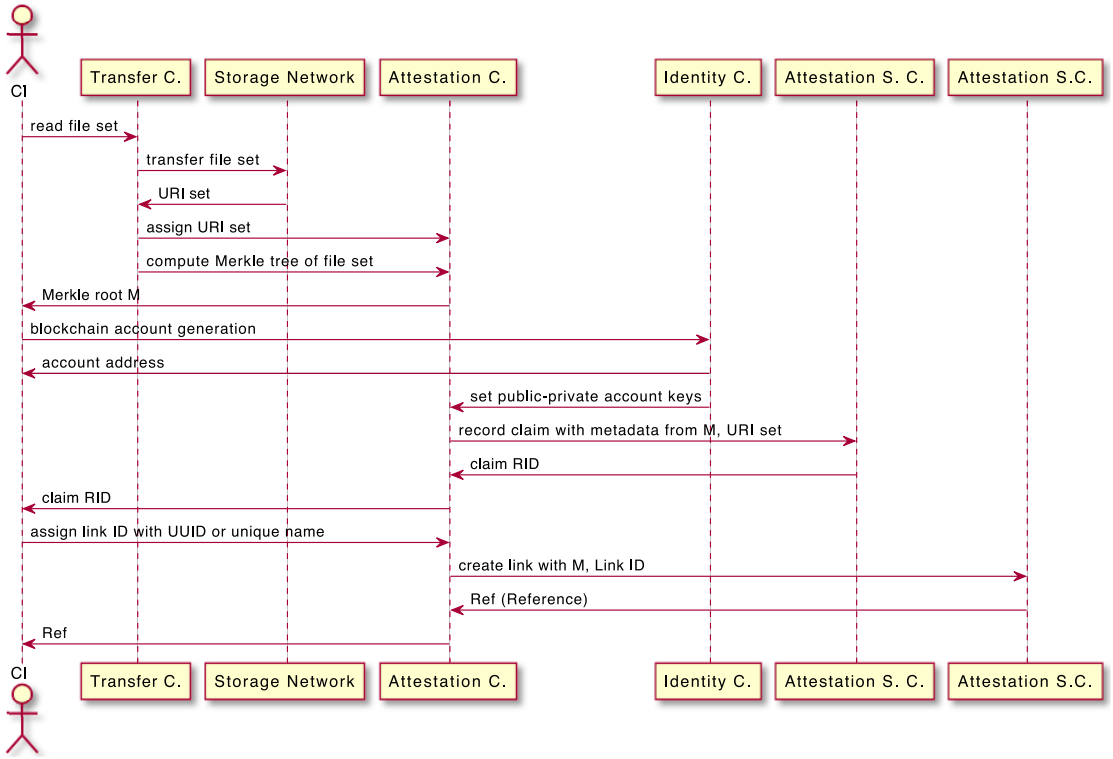

FIGURE 3. UML sequence diagram showing the distribution of files, the issuance of a claim and the creation of a link with the amplius prototype implementation. The following abbreviations are used. CI: Claim Issuer, C.: Component, S.C.: Smart Contract, M: Merkle Root, RID: Record ID (of a claim), Ref: Reference (of a link).

distribution and validation of certifications must be established in two scenarios. Either certification documents are transferred point-to-point in the form of files, e.g. through e-mail, or made available through one-to-many distribution such as on web portals where certification files are stored and validated. In both scenarios, existing methods possess limits in that the validations are source-based (c.f. Section II-D). In these scenarios, the certification files and a mechanism establishing validity have to be available. For example, the digital signatures and pubic keys of the senders of e-mails with attached certifications must be validated in the point-to-point case, e.g. with public key infrastructures using key servers, even when certifications are forwarded to a third party. A typical one-to-many distribution will require the server and web-service to continuously serve or validate certifications. No guarantees exist for the validation to be always available and not change validation algorithms. For example, a company issuing industry certifications to professionals faces the challenge of providing the server for an indefinite amount of time; professionals face the challenge of having to rely on the server without transparency. For higher-education institutions and students, these challenges exist as well for issuing digital degrees and other areas, for example, in the case of student transfers between universities. Students taking part in transfer programs in the EU today cannot rely on digital transcripts due to the lack of attestations, among other factors.[6] The resulting requirements encompass transparency and availability, and, in scenarios where an attestation is validated by multiple distributed parties, distribution and interoperability.

Blockchain-based attestation potentially can provide benefits in these scenarios since attestations of certification files can be validated through a smart contract, acting as an impartial third party. Projected benefits are higher transparency when recording and validating attestations, documenting the existence of certifications from specific issuer identities at the recorded timestamps, and preserving the attestation over the long term. Coupling file distribution with blockchain-based attestation aims at providing an access mechanism to the attestation file sets over time, such that a trusted method of access exists to certification files. This is achieved by providing certification files through integrity-secured URIs specified by the creators of attestations and for them leaving open the possibility of updating URIs over time.

In this scenario, the evaluation assumes the attestation and distribution of digitally issued university degrees in XML format. Two datasets of fictitious university degrees for 500 fictitious students have been generated, including data on students, courses, grades, and institutions. For these entities, ID and name attributes are described in the format of a domain-specific modeling language encoded in individual XML files. Accounting for the different scenarios of point-to-point transfers and one-to-many distributions, the two datasets contain fictitiously generated information on degrees in different file sets.

[6]https://europa.eu/youreurope/citizens/education/university/recognition/index_en.htm

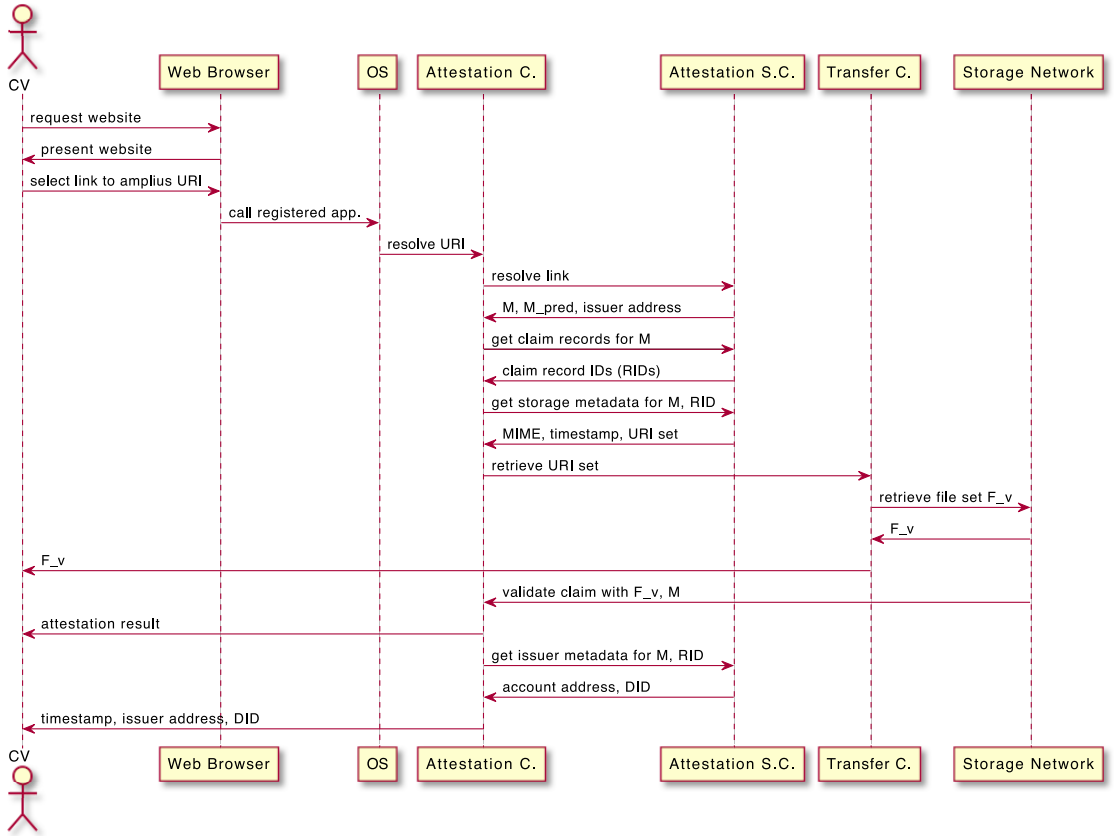

FIGURE 4. UML sequence diagram showing the retrieval of files from a link and validation of a claim with the amplius prototype implementation. The following abbreviations are used. CV: Claim Validator, OS: Operating System, app.: application, C.: Component, S.C.: Smart Contract, M: Merkle Root, pred.: predecessor, F_v: file set of validation, DID: decentralized identifier.

- (F1) The file set consists of 1 XML file representing the course certifications of students enrolled in 14 universities in Switzerland and is distributed and shared in a one-to-many fashion among universities, e.g. for student transfer programs. All data has been generated fictitiously.[7]
- (F2) The file set consists of 500 XML files representing the course certifications of students, where each XML file contains data for one student and is distributed in a point-to-point fashion among universities or students. All data has been generated fictitiously.[8]

Both file sets contain the same information and show the possibility of attesting files individually or in aggregate. The case of F2 might be compared with a typical issuance of certifications, e.g. issued by e-mail from universities to students. The file sets were used as input for the client software carrying out distributions and attestations.

B. SETUP

The setup for all measurements consisted of the following components required for attestation and distribution:

- Client software: the python-based client from the implemented software prototype amplius in version 0.2[9] (see Section III-C) with python 3.8.
- Blockchain node: the go ethereum (geth) blockchain node in version 1.10.5[10] in a fully-validating configuration with tracing and indexing of all transactions, and pruning of historical data. For the initial synchronization, the node was running for approximately 12 weeks on a machine with an AMD 3700X processor, 32 GB of RAM, and a Samsung 980 Pro NVMe SSD behind a 1 Gbit/s fiber internet connection. For the measurements involving local processing (Section IV-C2), the node was moved to a current laptop with an AMD 5700U processor, 16 GB of RAM, and a SK Hynix BC711 NVMe SSD. All other measurements involving blockchain data (Section IV-C1) were read and calculated from the blockchain data directly, independent form the hardware.
- Ethereum network: the mainnet of the public Ethereum blockchain in its 'Berlin' version, the current version as of July 2021.
- Attestation smart contract: the Solidity smart contract from the implemented software prototype amplius in version 0.1[11] (see Section III-C).
- Storage network: the distribution using IPFS, Git, and HTTP was performed with arbitrary nodes of the IPFS network, Git servers by Github.com, and an Nginx

[7]The dataset is available at https://github.com/fhaer/ampl-case-study-3.
[8]The dataset is available at https://github.com/fhaer/ampl-case-study-5.
[9]https://github.com/fhaer/amplius
[10]https://github.com/ethereum/go-ethereum/releases
[11]https://github.com/fhaer/amplius

```
Amplius is a user interface for creating and verifying attested multi-protocol links (AMPL). The prototype supports the transfer clients IPFS,
Git, and HTTP.

<command> is one of the following attestation, identity, or transfer commands.

Attestation Commands:
--attest [file]*                    create an identity if none is present, distribute all files, issue a content-based claim
--issue-claim [uri]*                issue a content-based claim for all files retrievable under the given URIs
--link=UUID <merkle-root> [p]       create a link with a new UUID v4 as ID pointing to claim <merkle-root>, parent link ID <p>
--link=<link-id> <merkle-root> [p]  create a link <link-id> pointing to the claim <merkle-root> with parent link ID <p>
--resolve=<link-id>                 resolve the claim Merkle root linked by <link-id>
--retrieve <merkle-root> <record-id> retrieve data of the claim identified by <merkle-root>, <record-id>
--validate-claim [file]*            validate a claim using the Merkle root of all files
--validate-claim=<record-id> [file]* validate a claim under the specified record ID using the Merkle root of all files
--validate-claim=<issuer> [file]*   validate claims issued by account address <issuer> with all given files

Identity Commands:
--eth-account-new                   create an identity for attestations
--eth-account-show                  show the current identity for attestations

Transfer Commands:
--distribute [file]*                distribute all files using all transfer clients with example repositories
--distribute=ipfs [file]*           add files to IPFS and pin at a remote node
--distribute=<GIT_URL> [file]*      commit and push all files with Git to <GIT_URL> starting with git or http and ending in .git
--distribute=<HTTP_URL> [file]*     send all files with HTTP PUT requests to <HTTP_URL> starting with http
--retrieve [URI]*                   retrieve all files from ipfs, git, or HTTP <URI>
```

FIGURE 5. Amplius prototype client command line user interface showing the implemented functions for each component.

HTTP server located in Switzerland. For the measurements involving local processing (Section IV-C2), an additional virtual machine running an Nginx HTTP server was used.

By design, the storage network involves various servers and protocols. The evaluation, therefore, involves measurements for providing access to the storage network through blockchain-based attestation, excluding the transfer and latency characteristics of servers of the storage network. For measurements involving the local processing of data obtained through requests (Section IV-C2), the requests were made to the virtual machine running locally instead of the storage network.

C. MEASUREMENTS

When recording claims and creating links, blockchain transactions induce cost and time expenditures for the issuers. For resolving links and validating claims, the retrieval of blockchain data and the processing of files on part of the validators require local processing time. This section describes the measurements taken for evaluating (1) the cost and time of blockchain transactions and (2) the overall local processing and request performance in comparison to web-requests using HTTP.

1) BLOCKCHAIN TRANSACTION COST AND TIME

The scalability problem of blockchains provokes the evaluation of transaction cost and time [56]. These measurements depend on the Ethereum blockchain, the attestation smart contract, and the input data encoded through the client software. The measurements were obtained from the resulting blockchain transactions, stored under the contract's address on Ethereum. For this reason, the results are visible in the Ethereum blockchain and can be seen, e.g. through a block explorer under the address of the smart contract.[12]

[12]E.g. https://etherscan.io/address/0x5627da24A01B5799AbC84300ACBf2A778933bEed

The following smart contract functions were executed and monitored:

- Registration functions: transactions calling the registerUriScheme, registerUriAuthority, and registerMimeType functions were called prior to the recording of claims. The functions register the existence of MIME types, URI schemes, and authorities used with http, git, and ipfs and assign them IDs for the recording of claims.
- Recording of claim: transactions calling the recordClaim function recorded attestation claims for file sets F1 and F2.
- Creation of link: transactions calling the link function were called for the creation of links to the recorded claims.

The smart contract was monitored for measuring transaction size and fees: the amount of data in the input field of the transaction and the gas usage from the execution on the Ethereum virtual machine. Measurements of confirmation times were derived from the blockchain. For the deployment of the smart contract, the gas usage and transaction cost of the contract creation were taken into account as a one-time expense.

2) PROCESSING AND REQUEST PERFORMANCE

While links and claims are registered on the blockchain, their resolving and validation occur through local reading and processing of the blockchain data. In comparison to the common case of retrieving files through HTTP web-requests, the client is required to perform additional computations locally. The client will first resolve a link and obtain claim records using blockchain data, retrieve the corresponding files through HTTP, and validate the claim given the file set (c.f. Figure 4). As preconditions, the publication of the files on the web and, after an undefined period of time, the issuance of a claim record and a link are carried out (c.f. Figure 3). The local processing required for these operations occurs only once and is independent of the client and the source of the files. In order to account for these aspects of processing in comparison

to web-requests, the time of the following operations was measured with the prototype implementation:

- HTTP PUT Request: the transfer time from the start of each request to its completion as a baseline for comparison.
- Recording of claim and creation of link: the processing time of all local computations without blockchain transactions, including the computation of Merkle trees from files and the link ID assignment of random UUIDs.
- Retrieval of link and claim: the processing time of all local computations without blockchain transactions, involving querying the blockchain data locally with the attestation smart contract for the link and the claim records.
- HTTP GET Request: the transfer time from the start of each request to its completion as a baseline for comparison.
- Validation of claim: the processing time of all local computations without blockchain transactions, involving the computation of Merkle trees from files.

These measurements were conducted for the file sets F1 with 1 file, F2 with 500 files, and 499 subsets of F2 with different numbers of files ranging from 1 file to 499 files. All measurements of requests serve only as a basis for comparison. They were made to a locally running virtual machine for excluding transfer and latency characteristics of remote servers. The time measurements were taken with the Python function time.perf_counter_ns().[13]

D. RESULTS

The following three subsections discuss (1) results for measurements related to blockchain transactions, (2) their extrapolation over time, and (3) results of processing and request performance measurements.

1) BLOCKCHAIN TRANSACTION COST AND TIME

Table 1 lists the results for each of the functions in the scenarios of both file sets. The size and cost per transaction do not differ between F1 and F2 and are discussed in the following.

TABLE 1. Results for smart contract functions in both scenarios involving file sets F1 and F2.

	Size in Bytes	Units of Gas	Cost in Ether
Registration functions	216	383520	1.3806720E-2
Recording of claim	708	217557	0.7832052E-2
Creation of link	100	69493	0.2501748E-2
Total	1024	670570	2.4140520E-2

For the registration functions, the size of transactions is 216 bytes. These transactions are required only once per registered MIME type, URI scheme, and authority. In the example, the registration amounts to 6 transactions, each with a size of 36 bytes, for registering the MIME type text/xml, the schemes IPFS, Git, and HTTP, and the URI authority for github.com, and an IP address. Each transaction of 36 bytes transfers 32 bytes of zero-padded data and a 4 byte function call. Transaction cost with a current gas price of 36 Gwei amounts to $36 * 383520 = 0.013806720$ Ether corresponding to 32.90 USD at an Ether price of 2383.24 USD.

Recording a claim involves the client-side computation of a Merkle tree for all files and the transfer of the Merkle root with the URI set within 1 transaction. The URI set references previously registered schemes and authorities through identifiers. Both the Merkle root and the identifiers possess a constant length for an arbitrary number of files stored as part of the Merkle tree. For this reason, the size and cost lead to the same result for F1 and F2. The per-transaction size amounts to 708 bytes transferring 704 bytes of zero-padded data and the 4 byte function call. Transaction cost with a current gas price of 36 Gwei amounts to $36 * 217557 = 0.007832052$ Ether corresponding to 18.67 USD at an Ether price of 2383.24 USD.

Creating a link for an existing claim results in 1 transaction containing a link ID as UUID or name, the Merkle root of the claim, and the previous Merkle root. The per-transaction size amounts to 100 bytes transferring 96 bytes belonging to a 32 byte link ID and two Merkle root values of the same size. Both Merkle root values are always transmitted even if no previous value exists. Transaction cost with a current gas price of 36 Gwei amounts to $36 * 69493 = 0.002501748$ Ether corresponding to 5.96 USD at an Ether price of 2383.24 USD.

The transaction size being independent of the number of files shows the advantage of using Merkle trees in this scenario. In the theoretical case of conducting 500 individual attestations, the total size and gas usage can be extrapolated from the per-transaction values to a size of $500 * 708 = 354000$ bytes and the gas usage of $500 * 217557 = 108778500$ units of gas. The total cost is 3.916026 Ether corresponding to 9332.83 USD with the assumed gas price of 36 Gwei and an Ether price of 2383.24 USD.

With regard to time, the confirmation of any transaction depends on it being included in a block by a miner on the Ethereum network such that measurements of few individual transactions are not meaningful. Furthermore, in times of high network utilization, transactions compete for the inclusion in the next block through gas prices. We revert to an assessment of time through typical confirmation times of the network. Confirmation times were monitored with the geth Ethereum node and aggregated to daily mean values with a relational database.

Table 2 shows the daily mean confirmation times in the time frame of the evaluation of blockchain transactions. The confirmation time depends on the number of consecutive blocks or confirmations, set by the client as a security parameter. When a transaction is made with the requirement of n confirmations, n consecutive blocks need to be observed before the data of the transaction can be accessed. With increasing n, the probability for double-spending attacks decreases in case re-organizations of the chain occur [57]. 10 to 12 confirmations might achieve a degree of security roughly similar to Bitcoin, assuming 6 confirmations on Bitcoin as a baseline for

[13] https://docs.python.org/3/library/time.html#time.perf_counter_ns

sufficient security [55], [57]. In practice, an acceptable number of confirmations might depend on the security requirements of the application.

Based on the 210 days of the time span, the daily mean ranges from 12.9 s to 13.6 s when waiting for 1 block as confirmation. Assuming 12 consecutive blocks are required for confirmation, the daily mean ranges from 156.2 s to 165.1 s. These relatively stable values can be assumed for transactions paying sufficient gas prices for being included in the next block.

TABLE 2. Daily mean of block confirmation times, January 1 2021 - July 29 2021.

Consecutive blocks	Min	Max	Mean
1	12.9 s	13.6 s	13.2 s
2	26.0 s	27.5 s	26.7 s
3	39.1 s	41.3 s	40.1 s
4	52.1 s	55.1 s	53.4 s
5	65.1 s	68.8 s	66.8 s
6	78.1 s	82.6 s	80.1 s
7	91.1 s	96.3 s	93.5 s
8	104.2 s	110.1 s	106.8 s
9	117.2 s	123.9 s	120.2 s
10	130.2 s	137.6 s	133.5 s
11	143.2 s	151.4 s	146.9 s
12	156.2 s	165.1 s	160.3 s
13	169.3 s	178.9 s	173.6 s
14	182.3 s	192.7 s	187.0 s

2) COST EXTRAPOLATION OVER TIME

While confirmation times can be assumed to be relatively stable, there exist multiple factors influencing the transaction cost over time. Major factors are the transaction size, the gas prices related to network utilization, and the Ether price. The transaction size depends on the implementation and can be considered invariant, with the exception of possible optimizations such as advanced compression algorithms.

For evaluating the possible effect of changes in gas prices and the Ether price, the cost of transactions recording claims and creating links has been extrapolated over time with historical data. In order to estimate transaction cost, gas prices were obtained from blocks between January 1 2017 and July 29 2021 using the geth ethereum node, and aggregated to daily mean values. Figure 6 shows the resulting cost based on these values and the transaction gas usage determined before. The Ether transaction cost for link creation and claim records is shown on the y-axis in comparison to the gas price on the secondary y-axes. Due to high network utilization in the last quarter of 2020 and the first two quarters of 2021, relatively high Ether transaction costs can be observed. When taking the price of Ether into account,[14] Figure 7 shows USD prices reaching close to 300 USD for the recording of claims before returning to tens of USD recently. Even though transaction cost is substantially reduced at the most recent point in time, it can be concluded that rising gas prices might again have a strong influence on transaction cost.

[14]Price data according to https://etherscan.io/chart/etherprice

3) PROCESSING AND REQUEST PERFORMANCE

Considering a scenario where clients are publishing and retrieving files on the web, the time required by the prototype has been measured, including the processing of attestations and links as well as HTTP requests. For an overview of the complete latency experienced by the user, the following discussion also includes estimations for blockchain transactions based on the confirmation times presented before. The results are subdivided into the time to publish files, the latency until blockchain transactions allow for the retrieval of files, and the time to retrieve files.

Starting with the publication of files, the header of Table 3 shows the required client-side operations in the order in which they occurred. Firstly, HTTP PUT requests for transferring individual files are carried out. Subsequently, local processing creates the claim, including the calculation of hash functions and the Merkle tree, as well as the link pointing to the claim.

TABLE 3. Time the prototype required to publish file sets F1 (n=1 file) and F2 (n=500 files) on the Web.

File Set	HTTP PUT	Create Claim	Create Link
F1	2.4 ms	3.0 ms	1.9 ms
F2	467.0 ms	4.0 ms	1.9 ms

The results for the two file sets, F1 with 1 file and F2 with 500 files, show the overhead for local processing in comparison to the time required for requests. The local processing of F1 causes a latency of 3.0 ms + 1.9 ms = 4.9 ms, increasing with F2 to 4.0 ms + 1.9 ms = 5.9 ms. In comparison to the time required for HTTP requests, the overhead is relatively small considering the size of the files, F1 with 556691 Bytes and F2 with 1213501 Bytes. However, while creating links is a constant-time operation, creating claims also depends on the number of files. For exploring this relationship, the measurements have also been performed with subsets of F2. In Table 4, the publication operations over the number of files from F2 subsets can be seen. The file size ranges between 2411 Bytes and 2434 Bytes with a mean of 2427.0 Bytes.

TABLE 4. Time the prototype required to publish n files in the subsets of file set F2 on the web.

n	HTTP PUT	Create Claim	Create Link
1	2.1 ms	3.4 ms	1.9 ms
10	11.2 ms	3.7 ms	1.7 ms
100	101.1 ms	3.9 ms	1.8 ms
200	212.7 ms	3.8 ms	1.9 ms
300	331.2 ms	3.8 ms	2.0 ms
400	467.3 ms	4.4 ms	2.0 ms

Before a file set becomes available for other clients to retrieve, the confirmation of the blockchain transactions induces latency. Separate transactions are made for the claim and the link, resulting in two transactions issued in

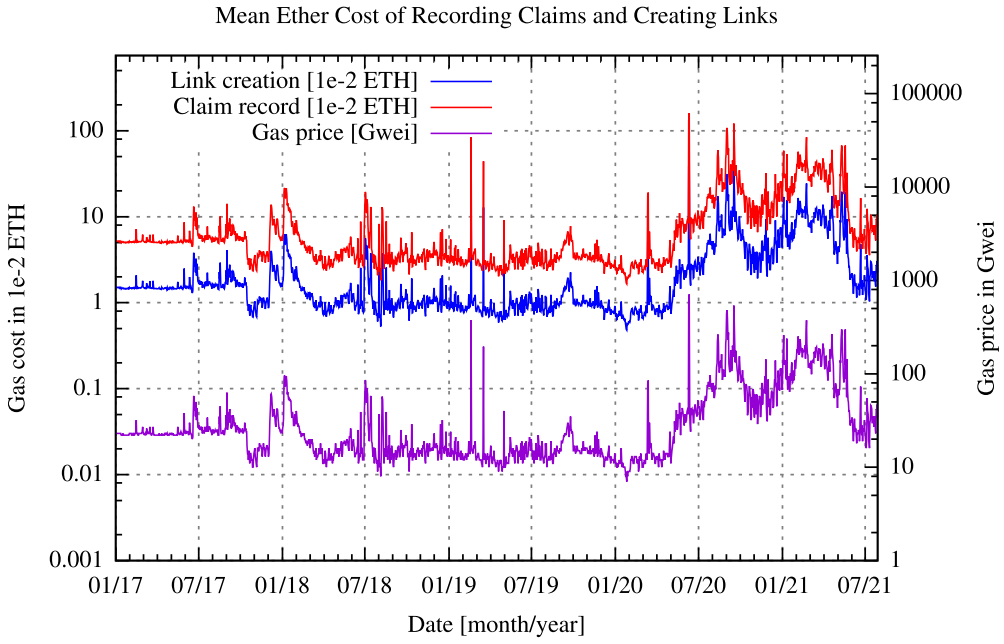

FIGURE 6. Ether transaction cost.

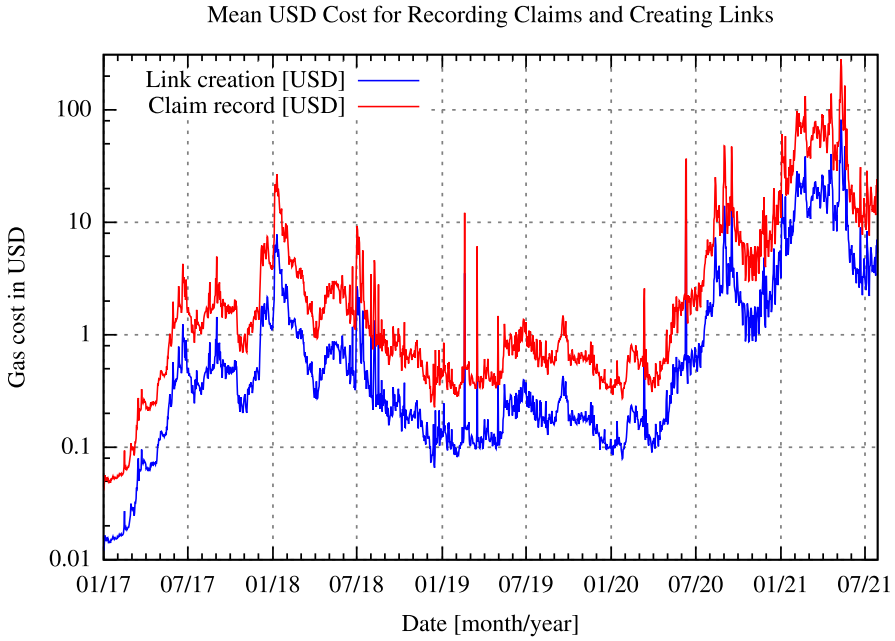

FIGURE 7. USD transaction cost.

sequence without delay. Typical confirmation times over the time span of the evaluation are estimated in Table 5, based on data from Table 2 with the assumption of 12 confirmations.

The results are the total confirmation times of both transactions for three cases. The first estimation assumes the two transactions are included in the same block, a likely scenario since both transactions are issued in sequence without delay. The second and third estimations assume the inclusion of the transactions within 2 blocks and 3 blocks, respectively. In the worst-case scenario, the blockchain-induced latency between the publication and the retrieval of a file set is 192.7 s. For making blockchain transactions and accessing the blockchain data, running a blockchain node in a fully-validating configuration is assumed for clients publishing or retrieving files (see Section IV-B).

The retrieval of files requires the client-side operations noted in the header of Table 6 in the given order. With a client in the possession of a link, local processing resolves the link and retrieves the corresponding claim from the local blockchain data. Individual HTTP GET requests for each file follow. Finally, local processing for the validation is carried out, including the calculation of hash functions and the Merkle tree.

TABLE 5. Estimated latency until blockchain transaction confirmations allow for the retrieval of files after publication.

Confirmation of two transactions	Min	Max	Mean
Same block	156.2 s	165.1 s	160.3 s
Within 2 blocks	169.3 s	178.9 s	173.6 s
Within 3 blocks	182.3 s	192.7 s	187.0 s

TABLE 6. Time the prototype required to retrieve file sets F1 (n=1 file) and F2 (n=500 files) on the web.

File Set	Resolve Link	Retrieve Claim	HTTP GET	Validate
F1	3.1 ms	11.9 ms	41.3 ms	3.2 ms
F2	2.1 ms	9.9 ms	513.8 ms	9.2 ms

For file set F1, the results show the overhead for local processing operations is 3.1 ms + 11.9 ms + 3.2 ms = 18.2 ms. For file set F2, an increase is observed in the validation time of 9.2 ms, resulting in a total overhead for local processing operations of 2.1 ms+9.9 ms+9.2 ms = 21.2 ms. Similar to the results for publishing files, the overhead is relatively small in comparison to the time required for HTTP requests. For taking also the number of files into account, Table 7 shows the results for the measurements performed with subsets of F2 containing different numbers of files.

TABLE 7. Time the prototype required to retrieve n files in the subsets of file set F2 on the web.

n	Resolve Link	Retrieve Claim	HTTP GET	Validate
1	2.0 ms	9.4 ms	1.3 ms	2.1 ms
10	2.0 ms	10.0 ms	10.8 ms	2.4 ms
100	2.2 ms	8.8 ms	109.7 ms	2.2 ms
200	2.0 ms	9.2 ms	216.3 ms	2.3 ms
300	2.0 ms	8.9 ms	353.8 ms	2.8 ms
400	2.1 ms	9.5 ms	461.8 ms	2.6 ms

Overall, the retrieval operations only carry out local processing with read-access to the blockchain in addition to HTTP-related operations. During or after the retrieval, no blockchain transactions take place.

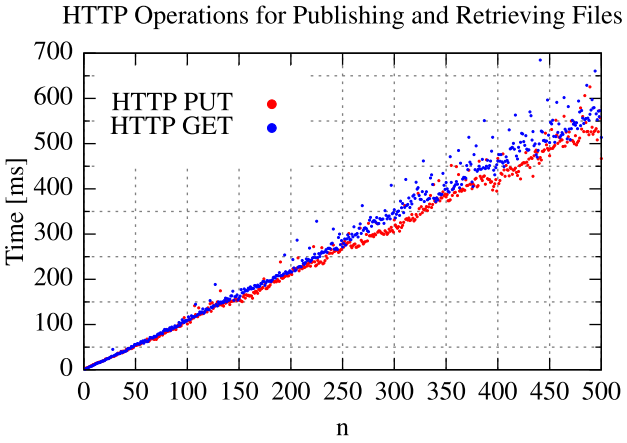

FIGURE 8. HTTP operations for publishing and retrieving file sets of size *n*.

In summary, Figure 8 visualizes all measurements collected for HTTP PUT and GET operations of the 1 to 500 files

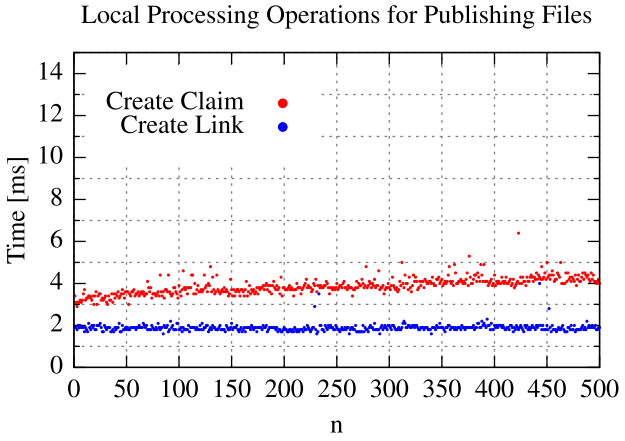

FIGURE 9. Local operations for publishing file sets of size *n*.

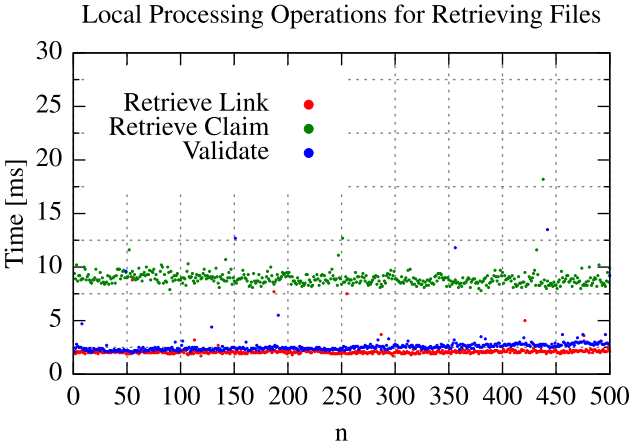

FIGURE 10. Local operations for retrieving file sets of size *n*.

in file set F2 and in its subsets. In comparison to the local computation operations, shown in Figure 9 and Figure 10 for publishing and retrieving files, the prior observations of a relatively low latency compared to HTTP operations can be seen. For the operations in Figure 9, the mean value for creating claims is 3.9 ms and the mean value for creating links is 1.9 ms. For the operations in Figure 10, the mean values for retrieving links, retrieving claims, and validations are 2.1 ms, 8.9 ms, and 2.6 ms, respectively.

V. DISCUSSION

The processing required for the publication and retrieval of files adds little overhead to the HTTP operations in the web-based scenario. However, before files become available for other clients to retrieve, the confirmation of blockchain transactions induces a latency of several minutes in the worst case. The approach is therefore limited to scenarios where such a latency is acceptable, such as typical web-based scenarios where newly published files are downloaded by other clients later on.

The additional cost of performing attestations when distributing files is on the order of tens of dollars at the most recent point in time and reached up to hundreds of dollars in the second quarter of 2021. Use cases are limited mostly by transaction cost, dependent on the network utilization visible

in gas prices, the price of Ether, and on the invariant transaction size. The application of blockchain-based attestation and distribution are therefore limited to scenarios where such an expenditure can be reasonably assumed. This might be the case for issuing degrees in special cases, e.g. the completion of study programs and issuance of high-value industry certifications. There might be other areas where the cost of individual attestations would not play a major role, such as the attestation of patents or the establishment of prior art in copyright law.

In the end, the high price of attestations is a symptom of rising prices for space in blocks on public blockchains. This effect can lead to blockchain technology being reserved only for high-value transaction scenarios, e.g. financial transactions for settlements using cryptocurrency or the attestation of legal documents. On the other hand, the continuous adoption in the direction of multiple blockchains and side-chains such as Polkadot,[15] Polygon,[16] and potential improvements in Ethereum 2.0[17] might lessen the network utilization in existing networks and might provide sufficient capacity overall. If transaction cost does not decline and remains at today's levels, the current cost of attestations on the order of tens of USD would be sufficient for the applicability of the approach in mid- to high-value certification scenarios.

VI. CONCLUSION AND OUTLOOK

In this paper, we described a novel approach for the decentralized attestation and distribution of information using blockchains. In contrast to previous approaches, it reverts to a multi-protocol concept, which permits to augment existing storage protocols such as Git or IPFS with blockchain-based attestations for verifying the authenticity and timestamps of the stored information. Through a first prototypical implementation the technical feasibility of the approach could be positively evaluated. Further, measurements of the performance of the approach showed a constant size of transactions for recording links and a moderate cost and completion time when reverting to the public Ethereum blockchain.

Future research will include in particular the extension of the approach towards other blockchain platforms. With the currently witnessed steep technological progress of blockchain platforms, it will be of interest to further study how higher transaction speeds and lower transaction costs will affect a more widespread adoption of the proposed approach. Similarly, the approach could be joined with blockchain-based mechanisms for processing the stored data, e.g. for reasoning over the content of data in a decentralized fashion to verify content-related properties.

REFERENCES

[1] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, "The world-wide web," *Commun. ACM*, vol. 37, no. 8, pp. 76–82, 1994, doi: 10.1145/179606.179671.

[2] R. Khare and S. Lawrence, *Upgrading to TLS Within*, document RFC 2817, 2000, doi: 10.17487/RFC2817.

[3] C. Hosmer, "Proving the integrity of digital evidence with time," *Int. J. Digit. Evidence*, vol. 1, no. 1, pp. 1–7, 2002.

[4] S. Bian, G. Shen, Z. Huang, Y. Yang, J. Li, and X. Zhang, "PABC: A patent application system based on blockchain," *IEEE Access*, vol. 9, pp. 4199–4210, 2021, doi: 10.1109/ACCESS.2020.3048004.

[5] M. Wilkinson, M. Dumontier, and I. J. Aalbersberg, "The FAIR guiding principles for scientific data management and stewardship," *Sci. Data*, vol. 3, no. 1, Dec. 2016, Art. no. 160018, doi: 10.1038/sdata.2016.18.

[6] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, "A survey of peer-to-peer storage techniques for distributed file systems," in *Proc. Int. Conf. Inf. Technol., Coding Comput. (ITCC)*, vol. 2, Apr. 2005, pp. 205–213, doi: 10.1109/ITCC.2005.42.

[7] L. Torvalds, J. Hamano, and J. Pearce. (2020). *Git User Manual*. Accessed: Nov. 26, 2021. [Online]. Available: https://github.com/git/git/blob/master/Documentation/user-manual.txt

[8] N. Paskin, "Digital object identifiers for scientific data," *Data Sci. J.*, vol. 4, pp. 12–20, 2005, doi: 10.2481/dsj.4.12.

[9] I. Peters, P. Kraker, E. Lex, C. Gumpenberger, and J. I. Gorraiz, "Zenodo in the spotlight of traditional and new metrics," *Frontiers Res. Metrics Anal.*, vol. 2, p. 13, Dec. 2017, doi: 10.3389/frma.2017.00013.

[10] F. Harer and H.-G. Fill, "Decentralized attestation of conceptual models using the ethereum blockchain," in *Proc. IEEE 21st Conf. Bus. Informat. (CBI)*, Jul. 2019, pp. 104–113, doi: 10.1109/CBI.2019.00019.

[11] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Nov. 26, 2021. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[12] V. Buterin, G. Wood, and J. Wilcke. (2014). *A Next-Generation Smart Contract and Decentralized Application Platform*. Accessed: Nov. 26, 2021. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[13] Ethereum Foundation. (2021). *Ethereum Whitepaper*. Accessed: Nov. 26, 2021. [Online]. Available: https://ethereum.org/en/whitepaper/

[14] A. Hafid, A. S. Hafid, and M. Samih, "Scaling blockchains: A comprehensive survey," *IEEE Access*, vol. 8, pp. 125244–125262, 2020, doi: 10.1109/ACCESS.2020.3007251.

[15] J. Benet and D. Dias. (2020). *IPFS Architecture Overview*. Accessed: Dec. 17, 2021. [Online]. Available: https://github.com/ipfs/specs/blob/master/ARCHITECTURE.md

[16] H. Huang, J. Lin, B. Zheng, Z. Zheng, and J. Bian, "When blockchain meets distributed file systems: An overview, challenges, and open issues," *IEEE Access*, vol. 8, pp. 50574–50586, 2020, doi: 10.1109/ACCESS.2020.2979881.

[17] Y. Psaras and D. Dias, "The interplanetary file system and the filecoin network," in *Proc. 50th Annu. IFIP Int. Conf. Dependable Syst. Netw.-Supplemental*, Jun. 2020, pp. 80–89, doi: 10.1109/DSN-S50200.2020.00043.

[18] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002, doi: 10.1145/514183.514185.

[19] J. F. Kurose and K. W. Ross, *Computing Networking: A Top-Down Approach*, 6th ed. Boston, MA, USA: Pearson, 2013.

[20] J. S. Sobolewski, "Cyclic redundancy check," in *Encyclopedia Computing Science*. Hoboken, NJ, USA: Wiley, 2003, pp. 476–479.

[21] L. Chi and X. Zhu, "Hashing techniques: A survey and taxonomy," *ACM Comput. Surv.*, vol. 50, no. 1, pp. 1–36, Jan. 2018, doi: 10.1145/3047307.

[22] Q. H. Dang, *Secure Hash Standard*. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2015. [Online]. Available: https://www.nist.gov/publications/secure-hash-standard, doi: 10.6028/NIST.FIPS.180-4.

[23] M. J. Dworkin, *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2015. [Online]. Available: https://csrc.nist.gov/publications/detail/fips/202/final, doi: 10.6028/NIST.FIPS.202.

[24] G. Wood. (2014). *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Accessed: Feb. 8, 2019.[Online]. Available: https://gavwood.com/paper.pdf

[25] R. G. Shirey, K. M. Hopkinson, K. E. Stewart, D. D. Hodson, and B. J. Borghetti, "Analysis of implementations to secure git for use as an encrypted distributed version control system," in *Proc. 48th Hawaii Int. Conf. Syst. Sci.*, Jan. 2015, pp. 5310–5319, doi: 10.1109/HICSS.2015.625.

[26] (2020). *Zenodo General Policies v1.0*. Accessed: Nov. 26, 2021. [Online]. Available: https://about.zenodo.org/policies/

[27] M. Belshe, R. Peon, and M. Thomson, *Hypertext Transfer Protocol Version*, document RFC 7540, 2015, doi: 10.17487/RFC7540.

[15] https://polkadot.network/

[16] https://polygon.technology/

[17] https://ethereum.org/en/eth2/

[28] M. Bishop, "Hypertext transfer protocol version 3 (HTTP/3)," Internet Engineering Task Force (IETF), Fremont, CA, USA, Tech. Rep. draft-ietf-quic-http-34, 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34
[29] W3C. (2001). *URIs, URLs and URNs: Clarifications and Recommendations 1.0*. Accessed: Feb. 28, 2019. [Online]. Available: https://www.w3.org/TR/uri-clarification/
[30] M. Dürst and M. Suignard, *Internationalized Resource Identifiers IRIs*, document RFC 3987, 2005, doi: 10.17487/RFC3987.
[31] P. Biswal and O. Gnawali, "Does QUIC make the web faster?" in *Proc. IEEE GLOBECOM*, Dec. 2016, pp. 1–6, doi: 10.1109/GLOCOM.2016.7841749.
[32] T. Kuhn and M. Dumontier, "Trusty URIs: Verifiable, immutable, and permanent digital artifacts for linked data," in *Proc. Semantic Trends Challenges*, V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin, S. Staab, and A. Tordai, Eds., 2014, pp. 395–410, doi: 10.1007/978-3-319-07443-6_27.
[33] A. R. Naik and B. N. Keshavamurthy, "Next level peer-to-peer overlay networks under high churns: A survey," *Peer-Peer Netw. Appl.*, vol. 13, no. 3, pp. 905–931, May 2020.
[34] B. Cohen, "Incentives build robustness in bittorrent," in *Proc. Workshop Econ. Peer-Peer Syst.*, vol. 6, 2003, pp. 68–72.
[35] C. Zhang, P. Dhungel, D. Wu, and K. W. Ross, "Unraveling the BitTorrent ecosystem," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 7, pp. 1164–1177, Jul. 2011, doi: 10.1109/TPDS.2010.123.
[36] V. Trón. (2021). *The Book of SWARM*. Accessed: Jul. 2, 2021. [Online]. Available: https://www.ethswarm.org/The-Book-of-Swarm.pdf
[37] A.-M. Kermarrec and M. van Steen, "Gossiping in distributed systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 2–7, Oct. 2007, doi: 10.1145/1317379.1317381.
[38] IPFS Project. (2021). *Content Addressing|IPFS Docs*. Accessed: Dec. 17, 2021. [Online]. Available: https://docs.ipfs.io/concepts/content-addressing/#identifier-formats
[39] A. Tenorio-Fornés, V. Jacynycz, D. Llop-Vila, A. Sánchez-Ruiz, and S. Hassan, "Towards a decentralized process for scientific publication and peer review using blockchain and IPFS," in *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, 2019, p. 10, doi: 10.24251/HICSS.2019.560.
[40] A. M. Antonopoulos and G. Wood, *Mastering Ethereum: Building Smart Contracts and Dapps*. Newton, MA, USA: O'Reilly Media, 2018.
[41] A. M. Antonopoulos, *Mastering Bitcoin: Programming the Open Blockchain*. Newton, MA, USA: O'Reilly Media, 2017.
[42] G. Coker, J. Guttman, P. Loscocco, J. Sheehy, and B. Sniffen, "Attestation: Evidence and Trust," in *Information Communications Security* (Lecture Notes in Computer Science), L. Chen, M. D. Ryan, and G. Wang, Eds. Berlin, Germany: Springer, 2008, pp. 1–18, doi: 10.1007/978-3-540-88625-9_1.
[43] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "ProvChain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2017, pp. 468–477, doi: 10.1109/CCGRID.2017.8.
[44] Y. Li, Y. Yu, R. Chen, X. Du, and M. Guizani, "IntegrityChain: Provable data possession for decentralized storage," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1205–1217, Jun. 2020, doi: 10.1109/JSAC.2020.2986664.
[45] S. Curty, H. Fill, R. S. Gonçalves, and M. A. Musen, "A webprotégé plugin for attesting to the provenance of ontologies on the ethereum blockchain," in *Proc. 20th Int. Semantic Web Conf.*, O. Seneviratne, C. Pesquita, J. Sequeda, and L. Etcheverry, Eds., Oct. 2980, pp. 1–5.
[46] C. Aebeloe, G. Montoya, and K. Hose, "ColChain: Collaborative linked data networks," in *Proc. Web Conf.*, New York, NY, USA, Apr. 2021, pp. 1385–1396, doi: 10.1145/3442381.3450037.
[47] H.-G. Fill and F. Haerer, "Knowledge blockchains: Applying blockchain technologies to enterprise modeling," in *Proc. 51st Hawaii Int. Conf. Syst. Sci.*, Waikoloa, HI, USA, 2018, pp. 4045–4054, doi: 10.24251/HICSS.2018.509.
[48] H.-G. Fill, "Applying the concept of knowledge blockchains to ontologies," in *Proc. Spring Symp. Mach. Learn. Knowl. Eng.*, Palo Alto, CA, USA, 2019, pp. 1–5.
[49] G. Caronni, "Walking the web of trust," in *Proc. IEEE 9th Int. Workshops Enabling Technol., Infrastruct. Collaborative Enterprises (WET ICE)*, Jun. 2000, pp. 153–158, doi: 10.1109/ENABL.2000.883720.
[50] N. Arndt, P. Naumann, N. Radtke, M. Martin, and E. Marx, "Decentralized collaborative knowledge management using git," *J. Web Semantics*, vol. 54, pp. 29–47, Jan. 2019, doi: 10.1016/j.websem.2018.08.002.
[51] F. Härer, "Decentralized business process modeling and instance tracking secured by a blockchain," in *Proc. 26th Eur. Conf. Inf. Syst.*, Portsmouth, U.K., 2018, pp. 1–5.
[52] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proc. Conf. Theory Appl. Cryptograph. Techn. Adv. Cryptol.* London, U.K.: Springer, 1988, pp. 369–378, doi: 10.1007/3-540-48184-2_32.
[53] Y. Liu, Q. Lu, H.-Y. Paik, and X. Xu, "Design patterns for blockchain-based self-sovereign identity," in *Proc. Eur. Conf. Pattern Lang. Programs*, Jul. 2020, pp. 1–14, doi: 10.1145/3424771.3424802.
[54] N. S. Borenstein and N. Freed, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodie*, document RFC 2045, 1996, doi: 10.17487/RFC2045.
[55] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, Oct. 2016, pp. 3–16, doi: 10.1145/2976749.2978341.
[56] S. Kim, Y. Kwon, and S. Cho, "A survey of scalability solutions on blockchain," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2018, pp. 1204–1207, doi: 10.1109/ICTC.2018.8539529.
[57] V. Buterin. (2015). *On Slow and Fast Block Times*. Accessed: Jan. 18, 2022. [Online]. Available: https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/

FELIX HÄRER was born in Erlangen, Germany, in 1987. He received the B.Sc., M.Sc., and Ph.D. degrees in information systems from the University of Bamberg, Germany, in 2012, 2014, 2019, respectively. From 2009 to 2012, he worked at Siemens Healthcare, Forchheim, Germany, as a Software and Testing Engineer. From 2014 to 2018, he was a Research Assistant at the System Development and Database Application (SEDA) Group, University of Bamberg. In 2018, he was a Research Assistant at the Digitalization and Information Systems (DIGITS) Group, University of Fribourg, Switzerland. Since 2020, he has been a Senior Research Assistant and lectures at the University of Fribourg. He has authored and coauthored several peer-reviewed publications, book chapters, and open-source software. His research interests include blockchains, decentralized and distributed systems, software engineering, and systems and software modeling.

HANS-GEORG FILL was born in Vienna, Austria, in 1978. He received the Diploma degree in international business administration and the Ph.D. and Habilitation degrees in business informatics from the University of Vienna, Austria, in 2002, 2006, and 2013, respectively. Since 2018, he has been a Full Professor with the Department of Informatics, University of Fribourg, Switzerland. He is the coauthor of several books, more than 70 peer-reviewed publications, and multiple open-source software systems. His research interests include enterprise modeling, blockchains, and augmented and virtual reality applications. He was a recipient of an Erwin-Schrödinger Scholarship for a research project at Stanford University, in 2010. He is a Co-Editor of the Department Enterprise Modeling and Enterprise Engineering of the journal *Business and Information Systems Engineering* and a Supporting Editor-in-Chief of the journal *Enterprise Modelling and Information Systems Architectures*—International Journal of Conceptual Modeling.