IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 3, 2021, accepted December 28, 2021, date of publication January 6, 2022, date of current version February 3, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3141014

Technological Foundations of Ontological Ecosystems on the 3rd Generation Blockchains

MIREK SOPEK[1], DOMINIK TOMASZUK[2], SZYMON GŁĄB[3], FILIP TUROBOŚ[3], IVO ZIELIŃSKI[1,4], DOMINIK KUZIŃSKI[1], RYSZARD OLEJNIK[1], PIOTR ŁUNIEWSKI[1], AND PRZEMYSŁAW GRĄDZKI[1]

[1]Research and Development Department, MakoLab SA, 91-062 Łódź, Poland
[2]Institute of Computer Science, University of Bialystok, 15-328 Białystok, Poland
[3]Institute of Mathematics, Łódź University of Technology, 93-005 Łódź, Poland
[4]Ivo Zieliński DLT Consulting, 90-368 Łódź, Poland

Corresponding author: Mirek Sopek (sopek@makolab.com)

This work was supported in part by the Ontochain (European Unions Horizon 2020 Research and Innovation Program) under Grant 957338, and in part by the GraphChain—a framework for on-chain data management for Ontochain under Grant 1477973.

ABSTRACT In this article we present the technological foundations on which an ecosystem of semantic data objects can be implemented on the latest Blockchain based systems. As the most important citizens among the semantic data objects are ontologies, the ecosystem is referred to as Ontospace. The foundations can be characterized by their architectural, cryptographic and transactional aspects. The architectural aspect borrows from the latest Layer-2 protocols of the 3rd generation blockchains and from the rules of Linked Data systems creation. The cryptographic aspect represents an original work that attempts to resolve the issue of efficient hashing of the graph data structures. The transactional aspect is concerned with the graph replication consistency, conditions for the direct access to graph data from the blockchain smart-contracts and with linkage between sidechains bearing semantic objects and the main network. The large parts of the work were implemented in the context of the Ontochain project – a part of the Next Generation Internet EU Initiative.

INDEX TERMS Blockchain, ontologies, semantic web, knowledge representation, RDF hashing algorithms, distributed databases.

I. INTRODUCTION

Blockchain technology [1], [2] has revolutionized the ways in which distributed databases are designed and implemented. Fully decentralized with cryptographically guaranteed immutability, consistence and persistence of data, Blockchain offers a general conceptual framework for the ultimately secured storage of data. However, as the technology was first implemented to facilitate a new kind of financial transactions and was later developed to enable general programming through the use of programs called smart-contracts which implemented generalized transactions, it was never meant to store large amounts of data or the data of the structure imposed by underlying data model, like relational databases, key-value or graph databases. If storing data on the Blockchain itself is an inevitable demand of a given project, the developers can do so by serializing the data elements and embed them into blocks of the chain. However, such an approach is inefficient. It does not facilitate efficient querying of data. For many popular Blockchains associated with traded cryptocurrencies it is also costly. The typical solution is to store data in some old-style centralized databases while keeping the data elements digests (hashes) on the Blockchain. To improve the storage model, instead of centralized database, distributed P2P data sharing networks have been invented, offering better data persistence.

This fundamental deficiency of standard Blockchains is particularly troublesome when the data to be stored is used to build knowledge representation systems, like Knowledge Graphs [3] or Ontologies [4]. What these systems desire the most is trust, the notion of shared truth and efficiency of knowledge discovery, extraction and consumption. When it comes to trust, the awareness of the need for mechanisms that makes it a fundamental feature of knowledge representation is present from the inception of the particular branch of the domain, namely – the Semantic Web [5]. The original Semantic Web Layer cake depicted the trust layer on top of the stack. However, there is still no unified and holistic trust

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

management model universally adopted even by the Semantic Web community [6].

On the other hand, trust in data consistency and immutability is the most important feature of Blockchains. Even in the trustless environment of the permissionless Blockchains, there are no reasons not to trust the data. The entire ecosystem of cryptocurrencies and decentralized financial products has been built on the notion of Blockchain trust.

The opportunity of using the Blockchain trust model to bring higher security and confidence in data motivated creators of solutions like: BigchainDB [7], ProvenDB [8], FlureeDB [9], Exonum [10] or ChainSQL [11]. The authors of this paper have created a working model of a system that directly addresses the challenge and brings Blockchain mechanisms to the RDF graph database [12]. This model is called GraphChain [13], and its first application was for enhancing trust of the digital identity system for legal entities [14].

Recently, the European Commission Next Generation Internet[1] initiative has launched a project called Ontochain.[2] The project aims at creation of ''Blockchain-based knowledge management solutions that address the challenge of secure and transparent knowledge management as well as service interoperability on the Internet''.[3] The authors of this paper have proposed a construction of a framework for on-chain data management for Ontochain based on the concept of aforementioned GraphChain solution.

The framework, depicted in the Figure 1, has been designed as an ecosystem of blockchains compliant with Blockchain Layer-2 protocol in which multiple sidechains coexist to provide services to different application domains. The Blockchains of the ecosystem can be closely coupled with graph databases, so that Blockchain state of the world includes the graph database state of the world. If the graph database is an RDF store, than the RDF named graphs correspond to the blocks of the Blockchain and effectively form a chain of distinguishable structures synchronized with the chain of blocks. The building blocks of the ecosystem called Ontospace (representing the entirety of Blockchains and semantic data pools) are:

- OntoSidechain – a single Blockchain of the Layer-2 protocol sidechain type,
- Ontonode – a single node of OntoSidechain,
- Ontopod – a part of Ontonode responsible for handling the semantic data chains of named RDF graphs – implemented with the use of the RDF graph databases (triplestores)
- Ontoshell – software modules for external communication for Ontonode (API & Linked Data via HTTP),
- OntoHub – a special OntoSidechain designed to store the most important top ontologies for the domain.

Designing and implementing such a system required meeting many challenges resulting from the architectural, cryptographic, and transactional aspects of the system. The architectural aspect concerned the implementation details of the Layer-2 protocol which was assumed to be the best foundation for the ecosystem construction and the design of components that allow for interaction with the system by standard web-based methods. We address this aspect in Section II. The cryptographic aspect concerned specific algorithms for calculating hash functions for graph structures and is addressed in Section III. Section IV elaborates on the mechanisms for transactionally correct synchronized updates of the Blockchain and graph database states. Section V is devoted to a discussion of the related work. The paper is concluded with a summary of the results obtained and with the plans for the further development.

a: CONTRIBUTIONS

The key contributions of our article can be summarized as follows:

1) Development of the architecture of the Blockchain based ecosystem for Ontologies and semantic data objects that preserves both Linked Data and Blockchain standards.
2) Exploration of the Layer-2 Blockchain protocols for the creation of trusted Knowledge Representation systems using multi-chain architectures.
3) Development of an innovative method for integrity proofs (hashing) for the named RDF graphs which includes the concept of vicious circle free RDF graphs.
4) A proposal for a modification of Blockchain (Ethereum) client which improves access to a database synchronised with Blockchain.

II. ARCHITECTURAL ASPECTS

A. GraphChain – THE FOUNDATION OF THE ONTOSPACE ECOSYSTEM

The definition and the first implementation of GraphChain, which forms the foundation on which Ontospace has been built was first proposed in 2018 [13].

GraphChain is a Blockchain solution where the fundamental data model is a collection of linked named RDF Graphs. GraphChain implements mechanisms typical to Blockchains such as data hashing, linking of named graphs into chains, replication of named graphs and achieving consensus on their content. The GraphChain network maintains a collection of RDF graphs in the databases of the network nodes forming the distributed system of chained RDF named graphs [15].

The fundamental advantage of such approach for the users is that they can work with the chained named graphs using standard tools developed in the domain of semantic web technology like SPARQL for quering [16], Linked Data mechanisms for accessing the nodes of the graphs [17], reasoners for ontologies [18] and many others – while benefiting from Blockchain mechanisms in their capacity to guarantee trust to the data.

The first implementation of GraphChains used the 1st generation Blockchain framework of Hyperledger Indy [13].

[1] https://www.ngi.eu/
[2] https://ontochain.ngi.eu/
[3] https://ontochain.ngi.eu/About

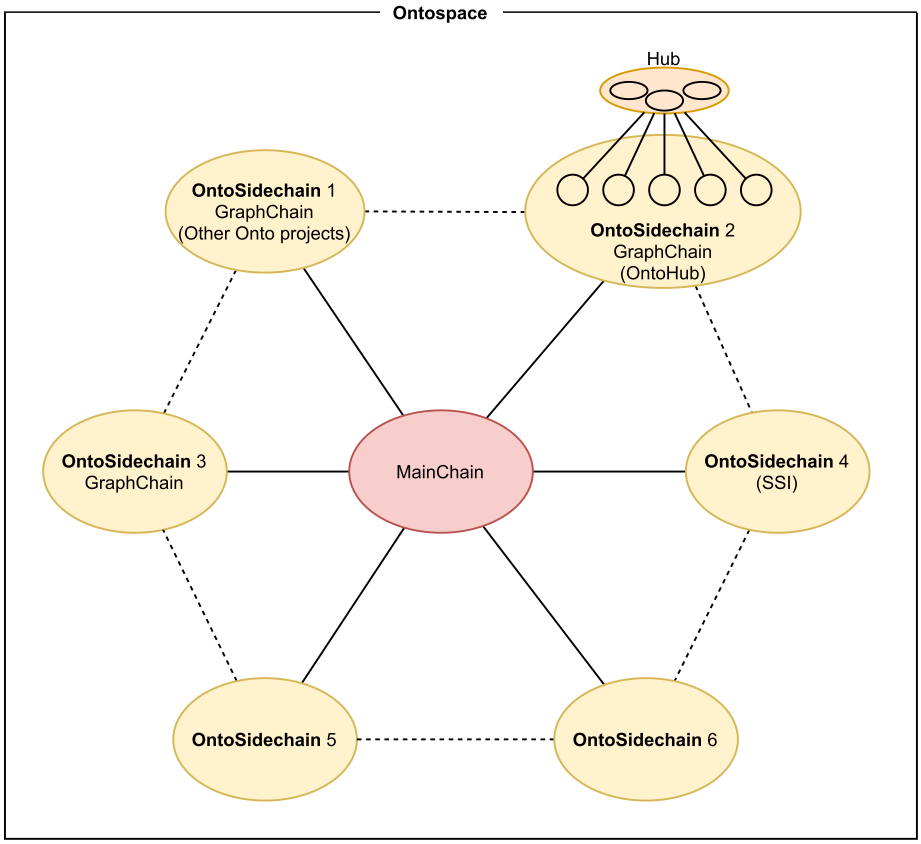


FIGURE 1. The Ontospace ecosystem.

While it was beneficial for the first application (the digital identity system for legal entities implemented for LEI.INFO portal[4] [14]), it was not rich enough for more sophisticated applications that required the capacity of a smart-contract based transaction. The work reported here implemented GraphChain using Ethereum based Layer-2 protocols, belonging to the family of the 3rd generation Blockchains.

B. 3rd GENERATION BLOCKCHAINS AND THE LAYER-2 PROTOCOL

When designing the architecture for Ontospace, we were motivated by the design principles of Layer-2 blockchain protocols and by principles of distributed databases. The Layer-2 blockchain protocols are overlying networks built on top of standard (Layer-1) blockchains, characterized by independent processing of transaction. From the standard, main blockchain perspective, these transactions are authenticated off-chain transactions, and the use of the main blockchain is reduced to the resolution of disputes. This allows for much faster transaction processing and the reduction of transaction cost. From the architectural perspective, they can be created in orthogonal way – as chains that operate independently and in parallel to the main blockchain network. The growth and popularity of various kinds of Layer-2 protocols (e.g. State Channels [19], Rollups [20], Plasma [21], ZK-STARKs [22], Commit Chains [23] and sidechains [24]) enabled creation of innovative financial services based on blockchains, known as DeFi (Decentralized Finance) [25].

However, while the main motivation behind the most of existing Layer-2 protocol designs is related to scalability issues related to the cryptoassets applications, what motivated authors of this paper was the need to design a system that combines blockchain design with the distributed knowledge representation system design based on the RDF semantic data objects. The design of such system assumes a specific modification of blockchain client to allow for a non-standard access to the RDF graph data. While such modification would not be recommended on the main network (Layer-1), it is possible on a Layer-2 sidechain, assuming that it allows the parent chain smart-contract to verify operations of the sidechains. Reasoning this way, we concluded that the application of some basic principles of Layer-2 sidechain protocols to the design of blockchain based knowledge representation systems, allows for the creation of the concept of an ecosystem, whose main purpose is to enable creation of trusted representation for solutions like Knowledge Graphs, Ontologies and semantic data sets – based on the RDF representation.

[4]https://lei.info/

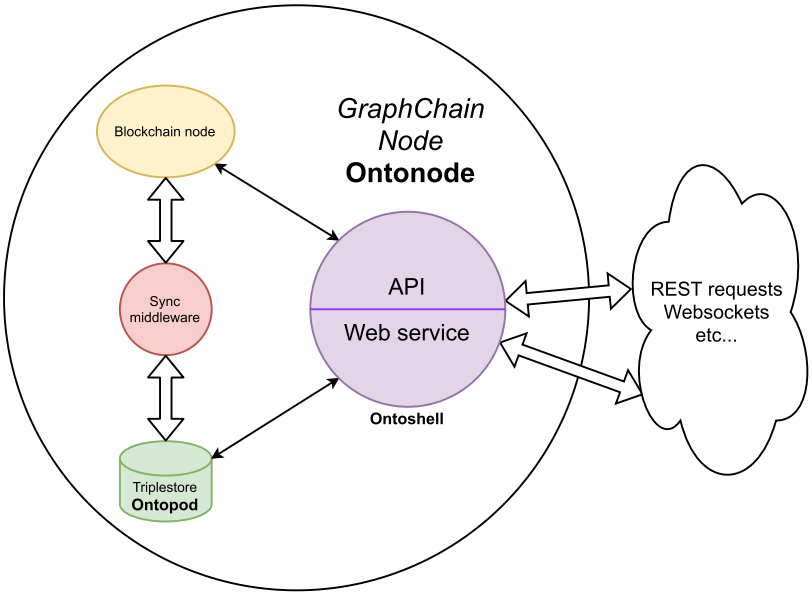


FIGURE 2. The design of Ontonode – a single node of the OntoSidechain.

The top level architecture of the Ontospace ecosystem has already been introduced in the previous section of the paper (see Figure 1), and was described in details in the previous paper [12]. However, for the completeness of explanation, let us review the ecosystem architecture. The ecosystem contains multiple sidechains, called OntoSidechains. Each sidechain of the ecosystem is a full-fledged blockchain network composed of Ontonodes, illustrated in the Figure 2. Ontonode is an integrated software component composed of Blockchain node, RDF Graph Database (Ontopod), synchronization middleware, and a set of modules responsible for external communication with the node (Ontoshell) which implements methods allowing for accessing the data in the RDF Graph Database.

Figure 3 presents another depiction of the ecosystem based on elements typical to Layer-2 protocol solutions. There is one particularly distinguished chain in the Ontospace – Ontohub. From the design perspective, it is identical to the other OntoSidechains, but its application is different. Designed as the primary distributed resource for the knowledge of the ecosystem, it serves as the repository of its most important (e.g. top-level) ontologies.

In accordance to the rules of Layer-2 protocols, the sidechains of Ontospace, the OntoSidechains are tethered (pegged) to the trusted parent network using mechanisms of Merkle Trees built with hashes of Ethereum transactions which include the RDF named graphs hashes. As this part of our work falls into transactional aspect of the system, it is described in Section IV. The section III is concerned with the cryptographic method for the RDF graph hashing.

C. ARCHITECTURE OF THE ACCESS TO THE RDF GRAPHS

The design of the OntoSidechain assumes that each node of the network (Ontonode) contains RDF graph database with identical content, i.e. the same chained sequence of named graphs. However, when building the web interface which allows for the interaction with the data according to the Linked Data principles, it is important to provide a unique end-point for the HTTP based access methods. This requirement was behind the access architecture presented in the Figure 4.

The access architecture assumes the presence of three layers:

1) The layer of distributed RDF graph databases replicated at every Ontonode of OntoSidechain.
2) The load balancer which distributes requests over the nodes of OntoSidechain
3) The web interface which implements SPARQL endpoint, REST API and provides user interface.

In the PoC phase of the aforementioned OntoChain project, the RDF databases were implemented using Blazegraph [18], for the load balancer Nginx web server [26] was selected and the web interface was implemented using Java Tomcat webserver [27].

One of the most important functionalities of the OntoSidechains is the ability to store ontologies. In a typical scenario the ontologies are processed in the OntoHub chain. For example, when a user plans to store ontology, the process illustrated in Figure 6 is initiated. It begins with the upload of the graph that represents an ontology. Firstly, the graph in a serialization of choice (Ontohub supports most of the RDF serializations) is posted to the Ontohub web interface, and it is then redirected by the load balancer to a selected Ontonode. Next, the service inserts the graph into the graph database (Ontopod) using standard SPARQL request or dedicated connector. When the insert succeeds, synchronization middleware detects that change and sends a new transaction to the blockchain node containing hash

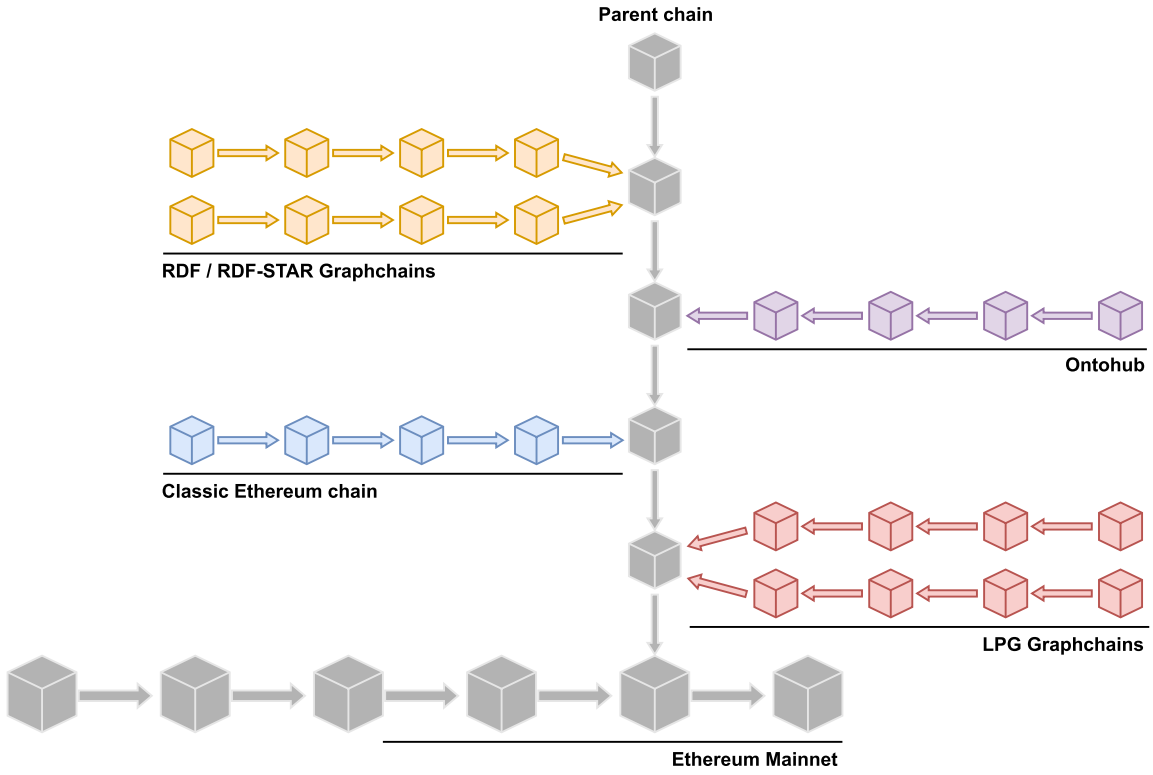


FIGURE 3. Ontospace as a collection of Layer-2 sidechains.

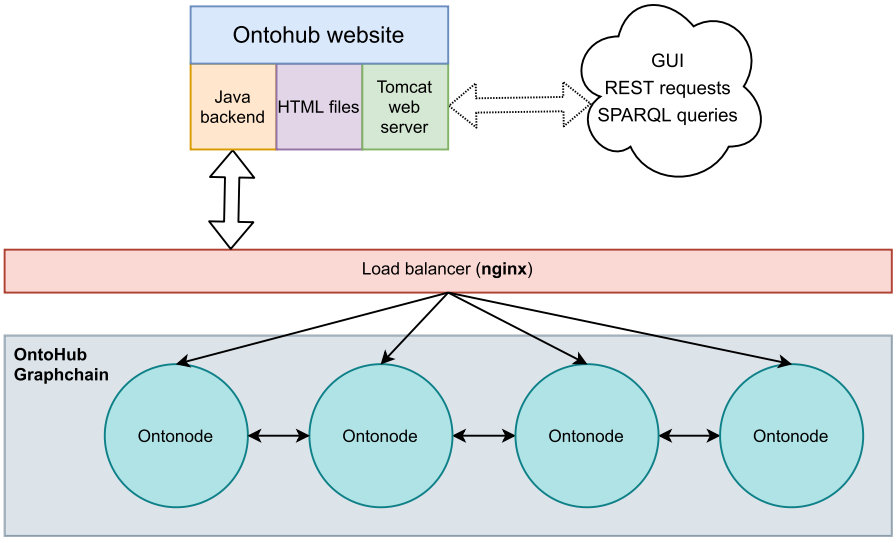


FIGURE 4. Access architecture to the OntoSidechain distributed RDF graph storage.

calculated from a new graph. After a successful transaction, the web service is informed about that fact and can return ''success'' response.

D. METAGRAPH

Because of the data immutability requirements, graphs stored in the system cannot be changed or deleted. Thus, when designing the graph database data management and the Ontoshell functionalities, a unique approach was required. Every time a graph is uploaded, Ontonode service has to check if the graph already exists, and if so, change the graph's name and store it without modifying previous versions. To keep track of the graph changes, the structure called Metagraph was proposed. The Metagraph is a special named

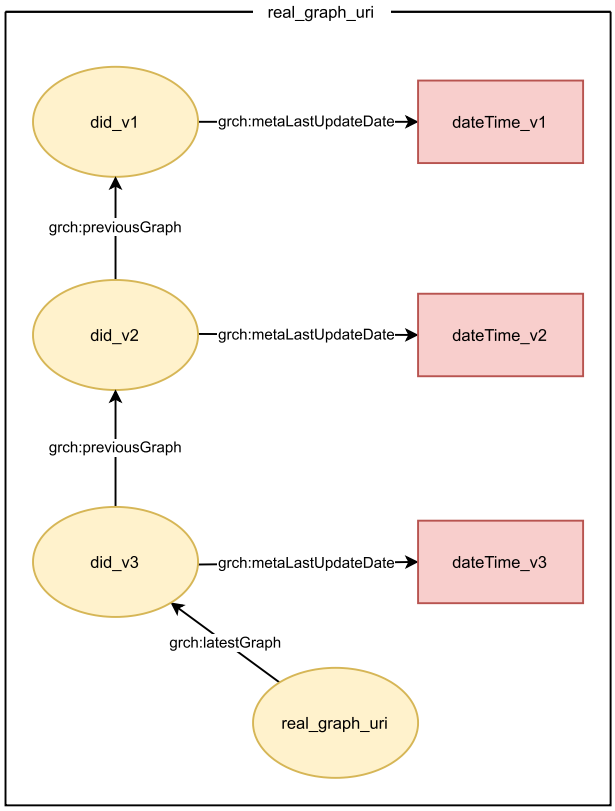


FIGURE 5. The metagraph.

RDF graph that has a name of an original graph uploaded by the user (further named `real_graph_uri`). Its content is a tree of graph version names with update dates for easy access to graph history, and with a pointer to the latest graph in sequence. The structure of Metagraph is depicted in Figure 5.

E. MODIFIED BLOCKCHAIN CLIENT

The key feature of the Ontonode is the tight coupling between the blockchain client and Ontopod – the graph database engine. In the case of Ethereum client, the coupling enables the graph data to be available almost as on-chain data, despite the fact that Ontopod object is external to the blockchain client. Typically in such a situation, third-party services, known as Oracles, connect smart-contracts with the external data [28]. In the design presented here, smart-contracts can interact with graph-data directly, without mediation of an Oracle. This was achieved by the creation of new EVM instructions (opcodes). In the PoC implementation with Hyperledger Besu, it was straightforward to create such new opcodes. At a minimum, a single read operation is needed – a function accepting the named graph ID to be fetched and returning the fetched graph. During the operation, a connection with the graph database is used to fetch the data. After the data is fetched, its integrity and authenticity is also verified. In the simplest case, it is not even necessary to modify the Solidity compiler to include the new opcodes. It is possible to embed the new operation in a singleton smart-contract that can act as a proxy, accessed using its address by all other smart-contracts requiring the graph database access. In a more sophisticated implementation, a modified Solidity compiler can translate an added language construct to the EVM opcode. We will discuss transactional aspects of such modification in the part IV of this paper.

III. CRYPTOGRAPHIC ASPECTS

An efficient mechanism for the RDF graphs integrity proofs, realized by cryptographic hashing is an essential part of the process of building a chained sequence of subgraphs (RDF named graphs [29]) that follows the creation of Blockchain blocks in OntoSideChains.

A. PRELIMINARIES

The graph data models are essentially different from the block data models used by most of Blockchain implementations and what is crucial in their handling is cryptographically safe and repeatable computation of the RDF hashes (digests). However, this seemingly simple task, is far from trivial [30] due to the following circumstances:

- The presence of the "blank nodes", i.e. the identifiers which are implementation dependent, so they may change while transferring the same graph between different RDF databases, or even between different access acts to the same database.
- No predefined order of RDF graph building blocks – the triples. They form an unordered set, causing the same graph serializations differences between different acts of access.
- When an RDF graph is serialized by different software routines, it can be encoded differently. As hashing is sensitive to encoding, it is subject to encoding-related problems.

There is also a more fundamental problem. The obvious goal of RDF graph hashing is to ensure that any two graphs that have the same hash are isomorphic. On the other hand, as the graph isomorphism is an NP-complete problem within the class of all possible RDF graphs, it seems infeasible to design a universal hashing algorithm that would be computationally efficient.

For that reason, when designing the Ontospace ecosystem, we have defined a subclass of RDF graphs for which we can construct a reliable, fast RDF hashing algorithm. We call this subclass "a vicious circle free class". The possibility of RDF graph to contain vicious circles is linked to the mechanism of blank nodes, i.e. the triples where subject or object are not expressed as global IRIs but as local addresses. Blank nodes represent unknown information which can represent something in the real world. Searching RDF graph for information can be compared to searching the encyclopedia, where unknown terms are explained by other terms, some of them unknown as well. Those unknown terms are analogous to the blank nodes. While searching them further in encyclopedia, we can find another unknown terms, etc. It would be a logical error of an encyclopedia, if there

is a circle in such a term explanation sequence. Imagine that the encyclopedia entry "vicious circle" refers to "circulus vitiosus", and "circulus vitiosus" refers back to "vicious circle". In this case, the process of finding the entry meaning would be its explanation (or it is rather a joke for those who know its meaning), but in most cases, such looping is a logical error known as a vicious circle. This reasoning demands a more formal analysis.

B. VICIOUS CIRCLE FREE GRAPHS

The IRIs used in RDF graphs are global identifiers that refer to any online resource, e.g. the IRI https://dbpedia.org/page/Lemon refers to a lemon fruit in DBpedia [31]. Literals are sets of lexical values. Blank nodes are locally-scoped identifiers for resources that are not otherwise named. Blank nodes cannot be referenced from outside of an RDF document. They can be bijectively relabelled without affecting the interpretation of the document. Accodring to [29], we define $\mathbf{I}$ as a set of all IRIs, $\mathbf{B}$ as a set of all blank nodes, and $\mathbf{L}$ as a set of all literals.

We also assume that $t = (s, p, o)$ is an RDF triple, where s is a subject, p is a predicate, and o is an object. The typical roles of elements s, p and o of RDF triple t are as follows:

- **subject** s is either an IRI or a blank node that refers to the primary resource described by t;
- **predicate** p is an IRI that identifies the relation between the subject and the object;
- **object** o is either a literal, an IRI or a blank node that fill the value of the relation.

According to [29], we define an RDF graph as a finite set of RDF triples, $G \subset (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$.

RDF graphs are isomorphic if they are the same up to bijective blank-node relabeling. Then, isomorphic RDF graphs have the same content. An RDF graph is a **set** of triples, so the triples in RDF dataset can be written in any order; moreover, blank nodes can be labelled arbitrarily. It is important to have an efficient algorithm finding whether two RDF graphs are isomorphic. A formal definition of RDF graphs isomorphism is the following.

Definition 1 (Isomorphism of RDF graphs): Let G and G' be two RDF graphs. Let B and B' be sets of all blank nodes used in G and G', respectively. We say that G and G' are isomorphic if there exists a bijection $\varphi : B \to B'$ such that for every RDF triple (s, p, o)

$$(s, p, o) \in G \iff (f(s), p, f(o)) \in G'$$

where $f(b) = \varphi(b)$ if $b \in B$ and $f(x) = x$ if x is IRI or literal.

C. EMBEDDING GRAPHS AND THE COMPLEXITY

Let $G = (V, A)$ be a finite directed graph. That is G consists of *vertices* $v \in V$ and *arcs*, that is ordered pairs $(v, w) \in A$. Two directed graphs G and H are isomorphic if there is a bijection $\psi : G \to H$ such that

$$(v, w) \text{ is an arc in } G \text{ if and only if } (\psi(v), \psi(w)) \text{ is an arc in } H.$$

We say that $v_1, \ldots, v_k$ is a *directed circle* in G if $(v_1, v_2), \ldots, (v_{k-1}, v_k)$ and (v_k, v_1) are arcs in G. A special kind of circle is a *loop*, that is a vertex $v \in G$ such that (v, v) is an arc. A *directed acyclic graph* is a directed graph with no directed circles.

Let $G = (V, A)$ be a directed finite graph. Let $\{v_1, v_2, \ldots, v_n\}$ be an enumeration of all vertices of G and fix n blank nodes $b_1, b_2, \ldots, b_n$ and one IRI u. For any arc $(v_i, v_j) \in A$, we define an RDF triple (b_i, u, b_j). Let us define an RDF graph $\bar{G}$ as a set of all those triples.

The RDF graph $\bar{G}$ looks very artificial. It consists entirely of blank nodes and such a document contains no information, but the structure of it includes G. It is defined against the RDF spirit. The definition of RDF graph is so general that it, unfortunately, includes such pathological examples. It is possible to show how to refine class of RDF graphs to omit such peculiar RDF documents, and still to have a large enough RDF graph class that covers most of interesting databases. Now, we will focus on showing that some kind of refining is really necessary to build a fast hashing algorithm. The following fact will allow us to prove that hashing algorithm problem is NP-complete in the class of *all* RDF graphs.

Theorem 1: Let G and H be two finite directed graphs. Then G and H are isomorphic if and only if $\bar{G}$ and $\bar{H}$ are isomorphic.

Proof: Let $\psi : G \to H$ be a graph isomorphism. Then, G and H have the same number of vertices, say n. Let $v_1, \ldots, v_n$ and $w_1, \ldots, w_n$ be enumeration of G and H, respectively, such that $w_i = \psi(v_i)$ for $i \leq n$. Further, let $b_1, \ldots, b_n$ and $b'_1, \ldots, b'_n$ be blank nodes used to define $\bar{G}$ and $\bar{H}$, respectively. Note that

$$(b_i, u, b_j) \in \bar{G} \iff (v_i, v_j) \text{ is an arc in } G \iff (w_i, w_j) \text{ is an arc in } H \iff (b'_i, u, b'_j) \in \bar{H}$$

which shows that $\bar{G}$ and $\bar{H}$ are isomorphic RDF graphs.

Now, suppose that $\bar{G}$ and $\bar{H}$ are isomorphic. It means that there is a bijection φ between $B = \{b_1, \ldots, b_n\}$ and $B' = \{b'_1, \ldots, b'_n\}$ such that for every RDF triple (b_i, u, b_j)

$$(b_i, u, b_j) \in \bar{G} \iff (\varphi(b_i), u, \varphi(b_j)) \in \bar{H}.$$

We define $\psi : G \to H$ by the formula: $\psi(v_i) = w_k$ if $\varphi(b_i) = b'_k$. Note that ψ is a bijection. Then

$$(v_i, v_j) \text{ is an arc in } G \iff (b_i, u, b_j) \in \bar{G}$$
$$(\varphi(b_i), u, \varphi(b_j)) \in \bar{H} \iff (\psi(v_i), \psi(v_j)) \text{ is an arc in } H.$$

Therefore G and H are isomorphic. ∎

It is well-known that a graph isomorphism problem is NP-complete for directed graphs. Theorem 1 along with the fact that there exist trivial polynomial-time conversions between graph G and corresponding RDF graph $\bar{G}$ implies that an RDF graph isomorphism problem is NP-complete as well. The embedding $G \mapsto \bar{G}$ of directed graphs to the class of RDF graphs that we have defined has a nice property prescribed in Theorem 1. Let us generalize this property to the

following definition. We say that the embedding $G \mapsto e(G)$ of finite directed graphs into RDF graphs is *proper* whenever

$$G \text{ and } H \text{ are isomorphic finite directed graphs} \iff e(G) \text{ and } e(H) \text{ are isomorphic RDF graphs.}$$

Let $\mathcal{C}$ be a subclass of all RDF graphs. The fact that isomorphism problem for $\mathcal{C}$ is not NP-complete implies that there is no proper embedding of finite directed graphs into $\mathcal{C}$.

D. VICIOUS CIRCLE FREE RDF GRAPH

Let's define a type of RDF bases in which no blank nodes reference sequence forms circles nor looping. We will call them vicious circle free RDF bases (or graphs).

Let G be an RDF graph. Let $B = \{b_1, \ldots, b_n\}$ be the set of all blank nodes in G. By $\mathcal{B}(G)$ we denote a directed graph whose set of vertices $V_{\mathcal{B}}(G)$ consists of all blank nodes of G and the set of arcs is given by

$$A_{\mathcal{B}}(G) := \{(b_i, b_j) : (b_i, u, b_j) \in G \text{ for some IRI } u\}.$$

We will call $\mathcal{B}(G) := (V_{\mathcal{B}}(G), A_{\mathcal{B}}(G))$ a *blank node subgraph of G*.

Definition 2 (Vicious Circle Free RDF Graph): We say that an RDF graph G is *vicious circle free* if its blank node subgraph $\mathcal{B}(G)$ is acyclic.

E. DETECTING RDFs WITH ACYCLIC $\mathcal{B}(G)$-GRAPHS

There are many known algorithms for cycle detection for directed graphs. Many of those are, in fact, adjusted version of methods used for topological sorting (see [32, Section 22.4] for details) of the graph. Here, we are going to present one of such methods, based on Kahn's algorithm, which works in $\mathcal{O}(V + E)$ time complexity [32, Exercise 22.4-5.] (see [33] for original paper).

Algorithm 1 presents a pseudocode, which happens to be a completely valid Python code that performs the detection for cycles in $\mathcal{B}(G)$ graph. For this method to be used properly, a `BG_Graph` has to be read previously while simultaneously keeping count on the structure of blank nodes.

```
1  Function cycleDetection(BG_graph):
2      queue ← new(Queue())
3      visited ← new(List())
4      for node in BG_graph do
5          node.temporary_degree ←
            node.blank_in_degree
           /* We keep count of unused edges
            for each node in a separate
            variable. */
6          if node.temporary_degree equals 0 then
7              add node to queue
8          end
9      end
       /* The main loop now commences.
       We 'peel' graph, taking one node
       at the time. Taking out a node
       with in-degree 0 temporarily
       removes the edges coming out of
       it, which decreases the in-degree
       of other nodes. We can only 'peel'
       vertices with in-degree equal to
       0.                                  */
10     while queue not empty do
11         node ← queue.pop()
12         for each neighbour in node.blank_neighbours
            do
13             Assign to L the number of edges from node
                to neighbour
14             neighbour.temporary_degree ←
                neighbour.temporary_degree - L
15             if neighbour.temporary_degree equals 0
                then
16                 add neighbour to queue      /* Mark
                    neighbour as suitable for
                    'peeling'. */
17             end
18         end
19         add node to visited        /* Mark node as
            visited. */
20     end
               /* If some vertices remain
       'unpeeled' then we know that graph
       contains a cycle. */
21     if len(visited) ≠ len(BG_graph) then
22         return True      /* Blank nodes graph
            contains cycle. */
23     else
24         return False  /* Blank nodes graph is
            acyclic. */
25     end
```

Algorithm 1: Detection for Cycles in $\mathcal{B}(G)$ Graph

F. INTERWOVEN HASH

Having a subclass of RDF graphs which do not have vicious circles defined, we can safely proceed with a proposal of hashing algorithm suitable for such class of RDF graphs.

We were inspired by the approach to the calculations of RDF Graph hash (digest), first presented in [13]. The authors of that approach defined the hash of the graph as a result of a specific summation of the hashes of all triples of the graph:

$$\mathcal{D}(S) = \bigodot_{i=1}^{N} h(serialisation(t_i))$$

We named this approach as "DotHash" algorithm. Its summation operation is associative and commutative and allows for the implementation of incremental algorithm in which the computation of the hash of the graph created by addition of new triples can be done by combining the hash of original graph and the sum of hashes of the added triples. An optimal (i.e. exhibiting a good compromise between performance and security) approach is based on the "AdHash" algorithm [34] and defines the specific

summation operation as a modulo operation with a suitably large value of the divisor. Incrementality of the calculations is of high importance for our application as it allows for a very efficient implementation in situations permitting new triples to be added to existing graphs. In addition to that, the method allows for highly efficient optimization. However, the method requires a very specific and non-generic approach when the graph contains blank nodes. In the original approach, its authors proposed a method where the blank nodes are labelled using statements like:

```
[ _bNode hasLabel L ]
```

Such labels are then used to rename the blank nodes during hash verification calculations. Of course, then, the calculation of hash will result in the same value. While this approach is practical, we did not find it generic enough and proposed the modified DotHash approach.

We named our approach ''Interwoven DotHash''. The fundamental feature of the ''Interwoven DotHash'' method is its ability to compute the graph hash without prior canonicalization of the entire graph nor a use of additional triples with ad-hoc labels. We will also use ''iHash'' acronym for the method.

The method we propose ignores the actual format of the blank nodes for the computation of hashes for the vicious circle free graphs while securing the essential effect of the blank nodes: their ability to what can be described as ''weaving'' multiple triples together.

We assume that the hash of a named graph is computed as the combining operation that is associative, commutative and supports incremental hashing of the graph.

As it was for DotHash, the combining operation can be implemented as a modulo (with a sufficiently large divisor) of the hashes of the triples. The hash of the triple is computed using the serialized triple form. Because the blank nodes are anonymous, the actual form and names do not matter. What is essential is the structure of the weaving of the triples together in the vicious circle free graphs.

Using such an approach, we propose the following algorithm for the calculation of the hash:

(1) If the triple does not contain a blank node, we compute its hash by applying SHA-256 algorithm [35] for its N-Triples serialized format [36].
(2) If the triple contains a blank node as its Subject, we compute its hash as the sum of SHA-256 results for the N-Triples serialized Predicate and Object and SHA-256 results for non-blank nodes of all those triples where the blank node appears in the Object nodes.
(3) If the triple contains a blank node as its Object, we compute its hash as the sum of SHA-256 results for the N-Triples serialized Subject and Predicate and SHA-256 results for non-blank nodes of all those triples where the blank node appears in the Subject nodes.
(4) If the triple contains blank nodes in both Subject & Object nodes, use the above rules twice, once for Subject, then for Objects.

```
1  Function hash(t):
2      if t.subject is blank node then
3          serialisation(t.subject) ← "Magic_S"
               /* Constant for blank node in
           subject. */
4      else
5          serialisation(t.subject) ← NTriples(t.subject)
6      end
7      if t.object is blank node then
8          serialisation(t.object) ← "Magic_O"
               /* Constant for blank node in
           object. */
9      else
10         serialisation(t.object) ← NTriples(t.object)
            /* If object is a literal we do
           additional normalization. */
11     end
12     serialisation(t.predicate) ← NTriples(t.predicate)
13     concatenation ←
        Concatenate(serialisation(t.subject),
        serialisation(t.predicate), serialisation(t.object))
14     return SHA-256(concatenation)
```

Algorithm 2: Basic Triple Hash Function

```
1  Function interwovenHash(G):
2      for t in G do
3          basicTripleHash ← hash(t)
                   /* Function from alg. 2 */
4          add to totalHash
                    /* Here we add hashes of
           adjacent triples to prevent
           ambiguity of blank nodes. */
5          if t.subject is blank node then
6              for all triples with t.subject in object
                position compute hash(t)
               add to totalHash
7          end
8          if t.object is blank node then
9              for all triples with t.object in subject
10              position compute hash(t)
               add to totalHash
11         end
12     end
13     return totalHash
14 end
```

Algorithm 3: Interwoven Hash Function

The iHash pseudo-code is presented in Algorithm 2 and Algorithm 3.

For practical application, we have implemented Interwoven Hash in Java, C#, Python, Javascript and Solidity.

IV. TRANSACTIONAL ASPECTS

The design choices we have made in the process of creation of Ontospace have impact on the transactional features of

the combined system of Blockchain and the RDF graph database. In this section, we address the most important processes of the combined system in its transactional behavior:

a. Replication of the RDF named graphs and assurance of consistency
b. Direct access to the RDF graph data using modified Ethereum client
c. Tethering of the Layer-2 OntoSidechains into the parent blockchains.

A. TRANSACTIONAL ASPECTS OF THE RDF GRAPHS REPLICATION

To ensure consistency of the synchronised replication of the RDF named graphs and the Blockchain replication and consensus mechanisms, an elaborated protocol has been proposed. The protocol assumes an interaction between the replication of graphs and the Blockchain internal replication mechanism through the use of specific smart-contracts. Such an approach helps in the preservation of the proper sequence of ''world state'' after transactions involving both graph databases and the blockchain.

To explain how the protocol works, it is best to follow the steps of the process from the user upload of a new graph to the insertion of the graph into the database local to each node of the system.

Figure 6 illustrates the process:

(1) User inserts a new graph through Rest API exposed by the Ontoshell component of Ontonode.
(2) Ontoshell services are activated:
  (2a) The Ontoshell service calculates iHash for the new graph.
  (2b) The graph is inserted into Ontopod (the graph database).
(3) The ''New graph'' method of the Ethereum smart-contract is executed, and the data about the new graph is stored in the blockchain, which is then propagated across the network through regular Blockchain mechanisms.
(4) The other nodes get info from their local Blockchain node about the new graph being available.
(5) The new graph info is retrieved from the smart-contract.
(6) The graph data is requested from the source Ontopod database.
(7) After validating graph data with iHash from the smart-contract, the graph is inserted into the local Ontopod instance.

1) GRAPH VERSION NAMES

An important factor of successful working of the process is the named graph identification scheme based on Internationalized Resource Identifier (IRI). The named graph version IRI is constructed in accordance with Digital Identifiers (DID) Syntax specification.[5] The adopted graph version IRI scheme is:

```
did:ihash:{IHASH}:{VERSION_TIME}
```

where:

- `IHASH` – iHash calculated for this graph written as hex-string (no caps)
- `VERSION_TIME` – update date as Unix time with milliseconds

We also used a regular expression to test the adopted IRI validity:

```
^did:ihash:[0-9a-f]{64}:[1-9][0-9]*$
```

An example of the graph version IRI in the metagraph from Figure 5 in N-triples format:

```
<did:ihash:548f41812b6c096bbf7fe5cfcb8
f9-a227cdf8fcca54ead07deee05658b134f3e:
1515-522403487>
```

The identifiers of graph versions have a specific structure that aids named graphs identification and management.

2) NEW GRAPH ADDITION

The graph database (Ontopod) endpoints are not directly exposed to external users. Instead, there is a special layer called Ontoshell, specifically designed to accept requests. The primary Ontoshell interface implements the SPARQL 1.1 Graph Store HTTP Protocol [37].

After a POST or PUT request, the RDF graph is not immediately stored in the graph database. Before that, iHash of that graph is computed and a new IRI for the current graph version is generated. Then, the graph is inserted into the graph database, the metagraph is created (or modified) and the smart-contract method is executed for adding the new graph info to the blockchain.

3) GRAPH DATA PROPAGATION

Blockchain data is propagated through the network in the standard way for the given Blockchain technology of choice (in the case described here – Ethereum). On every Ontonode there is a special task scheduled for polling blockchain ledger for new transactions with graph data. When a transaction with new graph arrives, Ontonode starts a graph database synchronization operation.

4) GRAPH DATABASE (Ontopod) UPDATE

Ontonode reads metagraph IRI (`real_graph_uri`), timestamp and iHash of the new graph content, assembles the graph version IRI from this info and sends a request to the source graph for actual graph data.

Then, a synchronizing code checks if the graph data iHash matches the one from the smart-contract, and if the metagraph content is the same as the local version (if present). If all checks are successful, the new graph is inserted into local

[5]https://w3c.github.io/did-core/#did-syntax

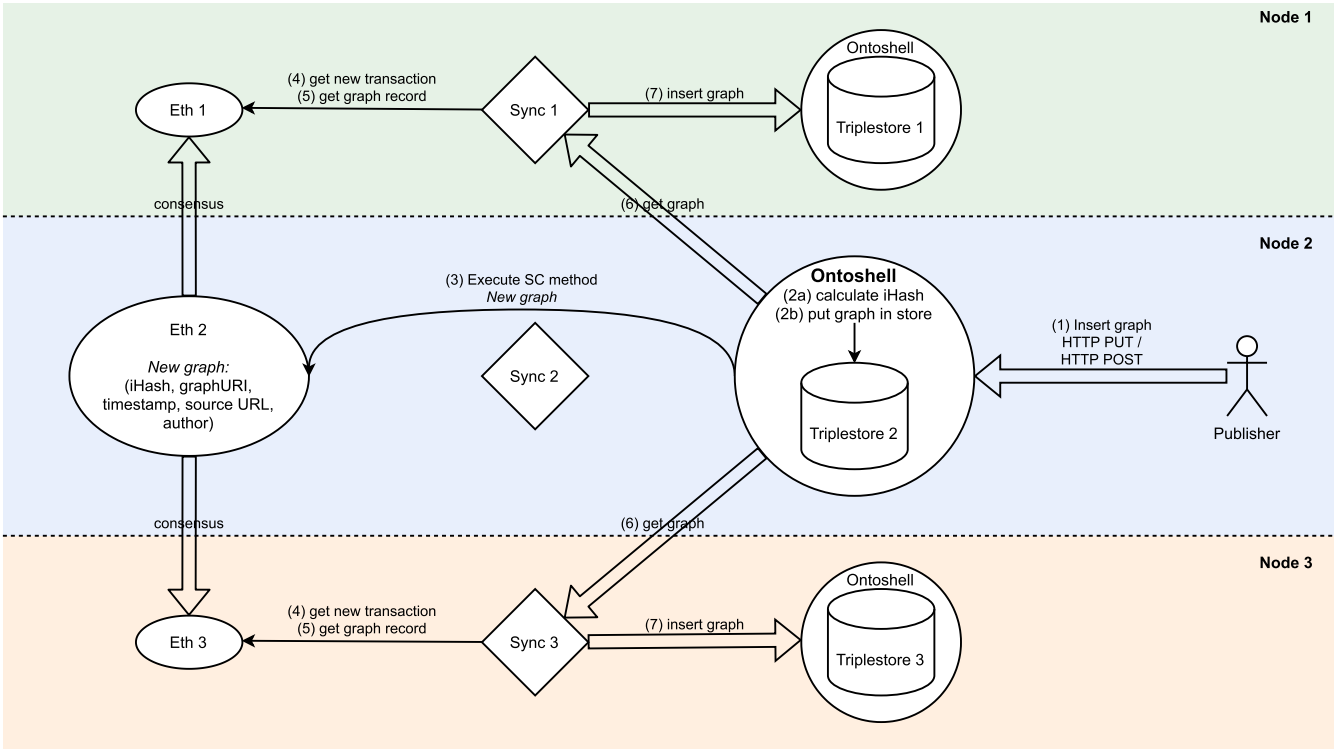


FIGURE 6. **Ontochain synchronization overview.**

Ontopod, the metagraph is updated and the node status is synchronized.

5) ADDING NEW OntoSidechain NODE

When adding a new node to the chain, or when restarting it after a longer pause, the synchronization mechanism is pretty much the same as for the working node. The graph records are downloaded in batches asynchronously. There is a separate smart-contract on the blockchain for storing indices of the last graph record synchronized for each node and also for the URLs of the nodes.

6) SMART-CONTRACTS

There are two smart contracts deployed on the OntoSidechain blockchain that are responsible for the graph synchronization mechanism: `GraphEventStorage` and `NodeSynchronizationRegistry`.

The `GraphEventStorage` contract is responsible for storing information about the graphs added to the OntoSidechain. After adding a new graph to the database, the Ontonode must call the `addGraph` function to notify all other Ontonodes about the new graph. The data contained in the `GraphEvent` structure will then allow other Ontonodes to fetch the new graphs from their original source.

The role of `NodeSynchronizationRegistry` contract is to persist the information about nodes and about Ontonode graph synchronization progress, i.e. how many graph events have been processed from the `GraphEventStorage` contract. Its most important field is `nodeSynchronizationProgress` which is a counter of the synchronized graphs.

If `nodeSynchronizationProgress` is equal to `GraphEventStorage.graphEventCount` for a given `nodeId`, it signifies that the Ontonode's graph database is up to date. If `nodeSynchronizationProgress` is less than `GraphEventStorage.graphEventCount` for a given `nodeId`, it signifies that the Ontonode's graph database needs to fetch new graphs from the source. If `nodeSynchronizationProgress` is zero for a given `nodeId`, it signifies that the Ontonode has not yet begun or has just started synchronizing the first batch.

The key feature of the synchronizer is the use of the smart-contract for monitoring its synchronization progress. After it finishes synchronizing a batch, it persists the index (N) of the last synchronized graph in the `NodeSynchronizationRegistry`, i.e. the index in the GraphEventStorage.graphEvents array. When the synchronization job runs again, it reads the value from the index $(N + 1)$. If $(N + 1)$ <= `GraphEventStorage.graphEventCount`, it starts the new batch.

The `NodeSynchronizationRegistry` smart-contract is also used to reinforce the replication mechanism. In the basic algorithm described in Subsection IV-A4, the synchronizer sends a request for the graph data

to the node that initially entered that graph into the system. If that node was unavailable for some reason, the synchronization process would be stopped. To prevent that from happening, the synchronizer uses the data from `NodeSynchronizationRegistry` to check which nodes have already got the particular graph (through `nodeSynchronizationProgress` field) and tries to send the request to them to fetch the data.

B. TRANSACTIONAL ASPECTS OF THE MODIFIED ETHEREUM CLIENT

To guarantee the direct access from smart-contracts to the RDF graphs, we have proposed a modification of the Blockchain client (we described it in the Subsection II-E of the Section II. However, such modification have important transactional ramifications as Ethereum is a ''transaction-based state machine'' where all transactions in the blockchain processed sequentially result in the same machine state, i.e. ''the version of the world of Ethereum''. All nodes of the network have to arrive at the same ''world'' state – otherwise, network participants would see different states (ex. account balances, elements of knowledge graphs) depending on which node they ask. To fulfill this requirement, all transaction processing from existing blocks needs to be deterministic. For this reason, in the standard implementation, the following operations, common to every programming language, cannot be performed by a smart contract:

- generating random numbers,
- directly calling an external APIs,
- external database operations.

The result of these operations can be unexpected and vary depending on the runtime and execution environment. API calls could succeed on some nodes, while failing on others – the returned values could also be different.

If a modified Ethereum client was to add an ability for smart contracts to access data outside of the internal state, then such a mechanism would need to satisfy the following requirements:

a: DATA IMMUTABILITY

Once accessed, the external data cannot be changed. As Ethereum transactions must be deterministic, the data access operations need to obey the same restriction. If the underlying external data would keep changing, then the same transactions would yield different states depending on when they were executed: for example, a new node synchronizing with the network processing old transactions would arrive at a different machine state than the nodes that processed the transactions as they were arriving. Henceforth, the underlying external database has to be append-only with no ability to delete old records. Moreover, it must be impossible to attempt to access data that does not exist, e.g., pass an ID of a non-existent record. If a past transaction failed to retrieve such a record and at some future point it came to existence, then divergent EVM states would occur – a state fork, as old transactions would start yielding different outcomes.

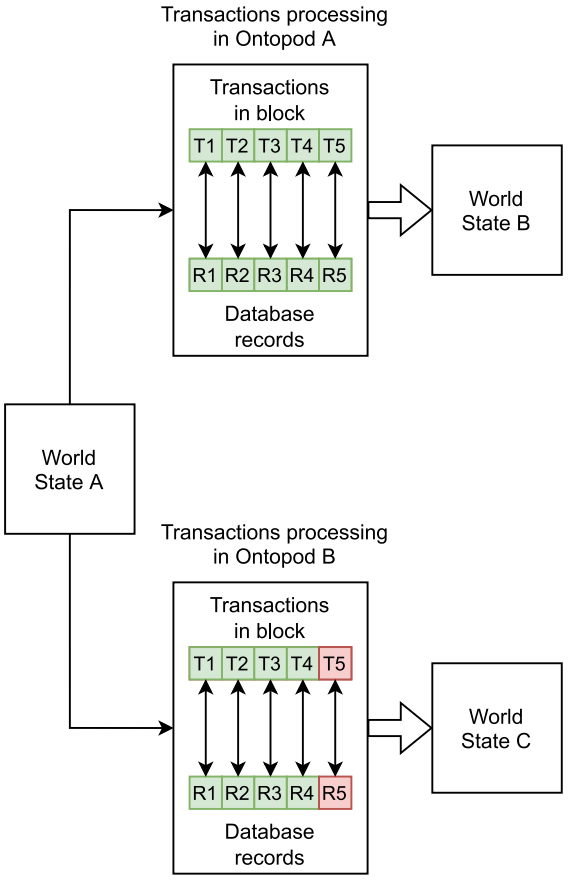


FIGURE 7. State synchronisation scenario.

b: EVENTUAL FINALITY

As Ethereum transactions processed by all the nodes in the network must yield the same machine state, it is obligatory that the external data provides time-critical guarantees of finality. Once the external calls are executed, there has to be a 100% certainty that they will yield the same results across the whole network.

In order to achieve that, a condition similar to this needs to be fulfilled: If a read from an external data source is executed and data older than X blocks is accessed, then the operation will be successful and return the same value for all nodes in the network.

The higher the X, the more time the external database network would have to synchronize new records. After X blocks have passed, a database read must return a record, i.e., all external database states older than X blocks must be final.

Guaranteeing this condition can sometimes be impossible in a complex distributed system, thus, it is crucial that proper mechanisms that prevent it from happening during the transaction processing phase are in place. Sometimes a database access operation fails unexpectedly due to network errors, hardware problems etc. If that is the case, the transaction cannot be accepted until it succeeds – if it does not do so, eventually it must be discarded. State synchronisation scenario is presented in Figure 7.

c: SECURITY GUARANTEES

As Ethereum networks and transaction processing are meant to be trust-less, the same requirement extends to the data provided by the external data source. Data providers can be incentivized to provide false data in order to extract value from the network, e.g., tokens locked in a smart contract. The data provision in a smart contract should be followed by checks guaranteeing that the data has not been tampered with. Ideally, it would be a verifiable cryptographic signature of the data creator.

d: DATA AVAILABILITY

In order for the newly connected Ethereum node to synchronize with the existing network, it will need to have access to a fully synchronized external data source to process all the transactions properly. As both external database and an Ethereum node could be starting in a desynchronized state, an additional external database will need to be provided for the needs of initial synchronization.

The efforts to meet these requirements have brought very good results. Using the PoC implementation of the system, we experimentally verified that the combined Blockchain and graph database system fulfills the requirements, while the access to the graph data was by the order of magnitude faster than with using Ethereum Oracles and an unmodified client [38]. All the aspects described in this section are valid independent of the consensus protocol used for a given sidechain. The architecture of our solution welcomes different consensus protocols depending on the target of the sidechain creation.

C. MECHANISMS FOR OntoSidechains TETHERING INTO THE PARENT CHAINS

Following Layer-2 protocols, Ontospace ecosystem demands sidechains to be tethered (or, in common parlance – pegged) to the mainnet (or to the parent net which then can be tethered to the mainnet). As the main goals of the Ontospace ecosystem are not just crypto-asset applications as it is the case with majority of Layer-2 blockchains, but they are related to the trusted knowledge representation, the mechanism we designed for the tethering is different from the standard Layer-2 mechanisms like rollups or state channels. What motivated our approach was the need for additional security of the data on Ontosidechains, gained from the linkage (tethering) to the mainnet. We have used Merkle Trees [39] for that purpose. Depending on the Ontosidechain activity (number of new blocks generated per unit of time), Every 2^N transactions of the OntoSidechain, a Merkle Tree is generated. In the leaves of the tree the mechanism stores both Blockchain transaction hash and the corresponding RDF named graph hash (iHash – Interwoven Hash described in the preceeding sections). Alternatively, only Blockchain transaction hash is stored, if the RDF named graph hash is already stored in the transaction. This is illustrated in Figure 8 and in Figure 9.

The roots of the Merkle Trees generated for the sidechain are stored in the transactions of the parent chain. Using

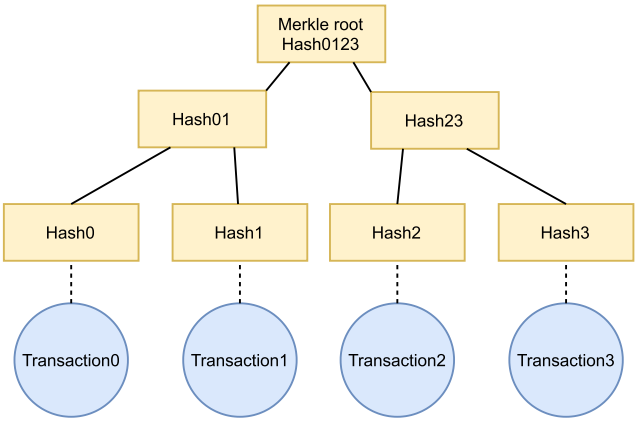


FIGURE 8. The Merkle Tree used for OntoSideChain tethering.

standard Merkle proof, every transaction can be audited and verified against the root of the Merkle Tree. Such an audit can be initiated on both the sidechain and on the mainnet, by a code specifically designed for the audits.

V. RELATED WORK

A. LAYER-2 APPROACHES

There are multiple types of Layer-2 implementations [40]–[45]. The first approach is Loopring [40] that runs as a public set of smart contracts responsible for trade and settlement, with an off-chain group of agents aggregating and communicating orders. Another proposal is AZTEC [41], which defines a set of zero-knowledge proofs that determine a confidential transaction protocol, designed for use within Blockchain protocols that support Turing-complete general-purpose computation. Another one proposal is Zecale [43] which is a general-purpose proof aggregator that uses a recursive composition of small arguments. Yet another approach is Hermes [42], which is a platform for trading sensor data, using distributed ledgers as intermediaries to add safeguards against malicious behavior. Other approaches are Raiden [44] and Lightning [45]. Both proposals operate on top of a blockchain and enable fast peer-to-peer transactions. They are based on state channels and the creation of communication channels between nodes out of the blockchain. Those networks are in charge of managing transactions between connected nodes, which reduces the main chain's workload.

On the other hand, Ethereum platform offers a few proposals for scaling [19]–[21]. These approaches are associated with a server or cluster of servers, each of which may be referred to as a node, operator, block producer, or similar term. State channels [19] utilize multisig contracts to enable participants to transact quickly and freely off-chain, then settle finality with the mainnet. Rollups [20] is another proposal that perform transaction execution outside Layer-1, and then the data is posted to Layer-1 where consensus is reached. There are two approaches based on rollups: ZK-rollups[6] and Optimistic rollups.[7] The first one generates

[6]https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/
[7]https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/optimistic_rollups/

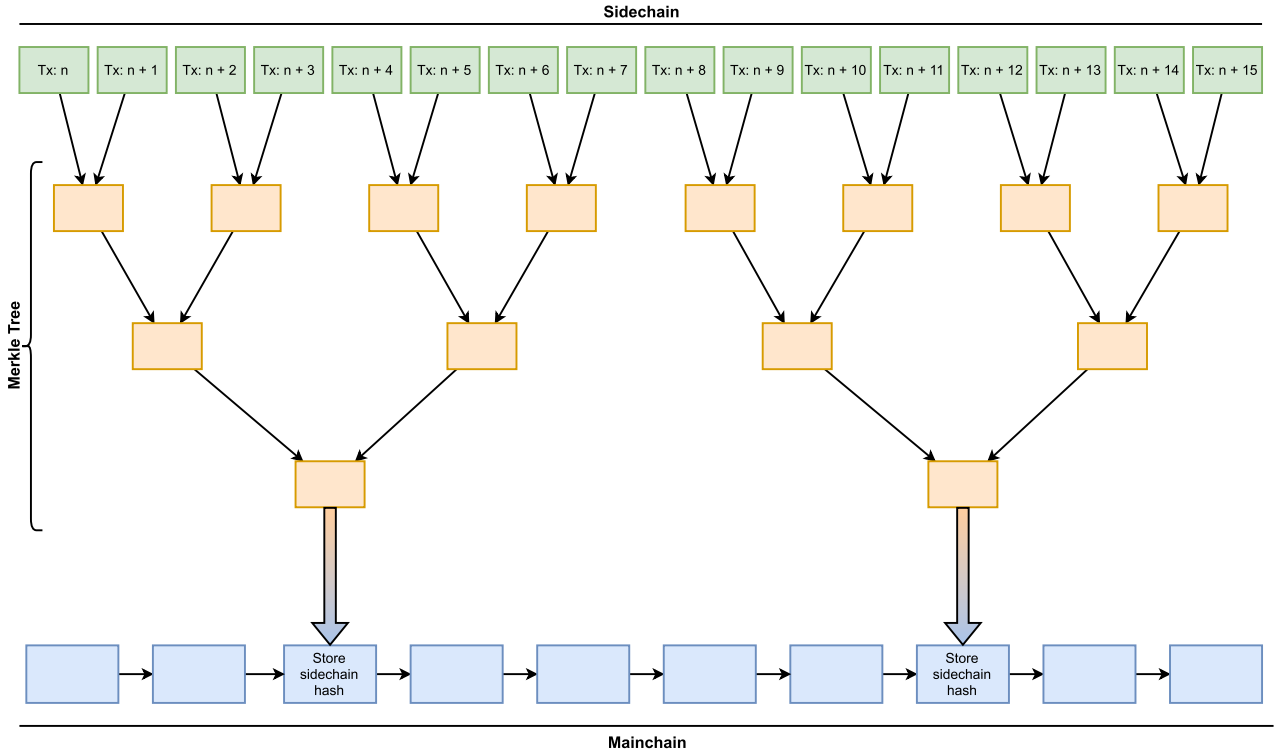


FIGURE 9. Pegged sidechain.

a cryptographic proof, known as a succinct non-interactive argument of knowledge. This is known as validity proof and is posted on the Layer-1. The second one offers improvements in scalability because, after a transaction, they propose the new state to the mainnet. The next approach is Plasma [21] that aims at extending the concept of sidechains, as a way to reduce the number of transactions to be processed by the Layer-1 Blockchain.

B. BLANK NODES AND GRAPH DIGESTS

Blank nodes are well-studied. There are papers covering their theory [46], semantics [47] and complexity [29],

Carroll [48] presents a method to canonicalise RDF graphs with blank nodes in such a manner that they could be digitally signed. The proposed method is based on writing it to N-Triples serialization, mapping all blank nodes to a global blank node, sorting the RDF triples lexically, and then relabelling the blank nodes. Another method to compute the digest of an RDF graph is proposed by Sayers and Karp [49], where there is an assumption that all blank node labels are fixed. Yet another method is proposed by Giereth [50]. In this paper, the author proposes to artificially add new triples to distinguish individual blank nodes in the encryption process. The next method is presented by Lantzaki *et al.* [51] and it is based on computing a signature for blank nodes based on the constant terms in their direct neighborhood. Yet another two algorithms are proposed by Hogan [52]. The first one computes an iso-canonical form and generates the same result for a pair of input RDF graphs if and only if they are isomorphic. The second one computes an equi-canonical form and generates the same result for pairs of simple-equivalent graphs.

C. MERKLE TREE-BASED GRAPH INTEGRITY

Crosby and Wallach [53] present History-Based Merkle Tree that is a tree-based history data structure for tamper-evident logging based on Merkle tree [39]. Position-aware Merkle tree is another method proposed by Mao *et al.* [54]. In this approach, each node in a Merkle tree can keep track of its relative position to its parent node. Yet another proposal is a Merklix tree [55]. It is a binary tree that has Merkle and radix tree features. Sutton and Samavi [56] proposed two methods: semantic-based approach and a structure-based approach. The first method leverages timestamps as an indexing key to construct a sorted Merkle tree variation. The second method utilizes the redundant structure of large RDF datasets to compress the dataset statements prior to generating a variation of a Merkle tree.

VI. CONCLUSION AND FUTURE WORK

Our work demonstrated a realistic possibility for the creation of Knowledge Representation system on Blockchain. This possibility stems from carefully designed synergy between RDF graph databases, Linked Data access methods and Blockchain functionalities that guarantee immutability, non-repudiation and decentralization of the Knowledge Representation realized by standard RDF graph databases. The work assumed meeting important challenges related to cryptographic methods adequate for the semantic objects expressed as RDF graphs, transactional requirements

resulting from a modified standard Blockchain client or architectural challenges resolved using 3rd generation Blockchain Layer-2 protocols.

As the theoretical work reported here was accompanied by a Proof-of-Concept kind of software development, we were able to positively verify the developed technological foundations.

The work is now in progress on a production grade system using the technologies described here, and on a possible application of our approach to the next generation of graph databases, i.e. the Property Graphs [57] which promise higher performance and new extended capabilities, but which require development of concepts like subgraphs, the property graphs hashing (integrity proofs) or the partial replication of such graphs.

REFERENCES

[1] T. Ahram, A. Sargolzaei, S. Sargolzaei, J. Daniels, and B. Amaba, "Blockchain technology innovations," in *Proc. IEEE Technol. Eng. Manage. Conf. (TEMSCON)*, Jun. 2017, pp. 137–141.

[2] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Bus. Inf. Syst. Eng.*, vol. 59, no. 3, pp. 183–187, Mar. 2017.

[3] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutiérrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, and A. Zimmermann, "Knowledge graphs," *ACM Comput. Surveys*, vol. 54, no. 4, p. 71/1–71/37, 2021.

[4] S. Staab and R. Studer, *Handbook on Ontologies*. Berlin, Germany: Springer, 2010.

[5] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Sci. Amer.*, vol. 284, no. 5, pp. 34–43, May 2001.

[6] S. R. Holagh and K. Mohebbi, "A glimpse of semantic web trust," *Soc. Netw. Appl. Sci.*, vol. 1, no. 12, pp. 1–10, Dec. 2019.

[7] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, "BigchainDB: A scalable blockchain database," whitepaper, 2016.

[8] ProvenDB. (2021). *ProvenDB—Your Trusted Data Integrity Solution*. [Online]. Available: https://www.provendb.com/

[9] B. Platz, A. Filipowski, and K. Doubleday, "FlureeDB: A practical decentralized database," whitepaper, 2017.

[10] Y. Yanovich, I. Ivashchenko, A. Ostrovsky, A. Shevchenko, and A. Sidorov, "Exonum: Byzantine fault tolerant protocol for blockchains," pp. 1–36, 2018.

[11] M. Muzammal, Q. Qu, B. Nasrulin, and A. Skovsgaard, "A blockchain database application platform," 2018, *arXiv:1808.05199*.

[12] D. Tomaszuk, D. Kuziński, M. Sopek, and B. Swiecicki, "A distributed graph data storage in ethereum ecosystem," in *Economics of Grids, Clouds, Systems, and Services*. Cham, Switzerland: Springer, 2021, pp. 223–231.

[13] M. Sopek, P. Gradzki, W. Kosowski, D. Kuziski, R. Trójczak, and R. Trypuz, "GraphChain: A distributed database with explicit semantics and chained RDF graphs," in *Proc. Companion Proc. Web Conf.* Geneva, Switzerland: Int. World Wide Web Conf. Steering Committee, 2018, pp. 1171–1178.

[14] M. Sopek, P. Grądzki, D. Kuziński, R. Trójczak, and R. Trypuz, "Legal entity identifier blockchained by a hyperledger Indy implementation of graphchain," in *Metadata and Semantic Research*. Cham, Switzerland: Springer, 2019, pp. 26–36.

[15] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler, "Named graphs, provenance and trust," in *Proc. 14th Int. Conf. World Wide Web*, New York, NY, USA, 2005, pp. 613–622.

[16] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of SPARQL," *ACM Trans. Database Syst.*, vol. 34, no. 3, pp. 1–45, Aug. 2009.

[17] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data—The story so far," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, 2009.

[18] S. Sakr, M. Wylot, R. Mutharaju, D. Le Phuoc, and I. Fundulaki, *Linked Data: Storing, Querying, and Reasoning*. Cham, Switzerland: Springer, 2018.

[19] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 949–966.

[20] V. Buterin. (2021). *An Incomplete Guide to Rollups*. [Online]. Available: https://vitalik.ca/general/2021/01/05/rollup.html

[21] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," whitepaper, 2017.

[22] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," IACR Cryptol. ePrint Arch., Tech. Rep., 2018, p. 46.

[23] R. Khalil, A. Zamyatin, G. Felley, P. Moreno-Sanchez, and A. Gervais, "Commit-chains: Secure, scalable off-chain payments," Cryptol. ePrint Arch., Tech. Rep. 2018/642, 2018.

[24] P. Robinson, R. Ramesh, and S. Johnson, "Atomic crosschain transactions for ethereum private sidechains," *Blockchain, Res. Appl.*, Sep. 2021, Art. no. 100030.

[25] D. A. Zetzsche, D. W. Arner, and R. P. Buckley, "Decentralized finance," *J. Financial Regulation*, vol. 6, no. 2, pp. 172–203, Sep. 2020.

[26] R. Soni, *Nginx*. Berkeley, CA, USA: Apress, 2016.

[27] A. Vukotic and J. Goodwill, *Apache Tomcat 7*. Berkeley, CA, USA: Apress, 2011.

[28] P. De Filippi, C. Wray, and G. Sileno, "Smart contracts," *Internet Policy Rev.*, vol. 10, no. 2, 2021.

[29] D. Tomaszuk and D. Hyland-Wood, "RDF 1.1: Knowledge representation and data integration language for the web," *Symmetry*, vol. 12, no. 1, p. 84, Jan. 2020.

[30] E. Höfig and I. Schieferdecker, "Hashing of RDF graphs and a solution to the blank node problem," in *Proc. URSW*, vol. 1259. Aachen, Germany: CEUR-WS.org, 2014, pp. 55–66.

[31] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "DBpedia: A nucleus for a web of open data," in *The Semantic Web*. Berlin, Germany: Springer, 2007, pp. 722–735.

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. Cambridge, MA, USA: MIT Press, 2009.

[33] A. B. Kahn, "Topological sorting of large networks," *Commun. ACM*, vol. 5, no. 11, pp. 558–562, 1962.

[34] M. Bellare and D. Micciancio, "A new paradigm for collision-free hashing: Incrementality at reduced cost," in *Advances in Cryptology*, W. Fumy, Ed. Berlin, Germany: Springer, 1997, pp. 163–192.

[35] (2015). *Fips Pub 180-4 Secure Hash Standard*. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[36] G. Carothers and A. Seaborne. (Feb. 2014). *RDF 1.1 N-Triples. W3C, Recommendation*. [Online]. Available: https://www.w3.org/TR/2014/REC-n-triples-20140225/

[37] C. Ogbuji. (Mar. 2013). *SPARQL 1.1 Graph Store HTTP Protocol. W3C, Recommendation*. [Online]. Available: https://www.w3.org/TR/2013/REC-sparql11-http-rdf-update-20130321/

[38] M. Sopek, D. Tomaszuk, S. Głąb, F. Turoboś, I. Zieliński, D. Kuziński, R. Olejnik, P. Łuniewski, and P. Grądzki. (2021). *Ontochain Foundations*. [Online]. Available: https://github.com/MakoLab/ontochain-foundations/

[39] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. IEEE Symp. Secur. Privacy*, Apr. 1980, p. 122.

[40] D. Wang, J. Zhou, A. Wang, and M. Finestone. *Loopring: A Decentralized Token Exchange Protocol*. [Online]. Available: https://github.com/Loopring/whitepaper/blob/master/en_whitepaper.pdf

[41] Z. J. Williamson. (2018). *The Aztec Protocol*. [Online]. Available: https://github.com/AztecProtocol/AZTEC

[42] P. Tzianos, G. Pipelidis, and N. Tsiamitros, "Hermes: An open and transparent marketplace for IoT sensor data over distributed ledgers," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, Seoul, South Korea, May 2019, pp. 167–170.

[43] A. Rondelet, "Zecale: Reconciling privacy and scalability on ethereum," 2020, *arXiv:2008.05958*.

[44] R. Homepage. (2021). *The Raiden Network*. [Online]. Available: https://raiden.network/

[45] J. Poon and T. Dryja, "The bitcoin lightning network," *Scalable O-Chain Instant Payments*, 2015.

[46] A. Mallea, M. Arenas, A. Hogan, and A. Polleres, "On blank nodes," in *The Semantic Web* (Lecture Notes in Computer Science), vol. 7031. Bonn, Germany: Springer, 2011, pp. 421–437.

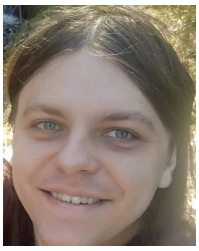
[47] A. Hogan, M. Arenas, A. Mallea, and A. Polleres, "Everything you always wanted to know about blank nodes," *J. Web Semantics*, vols. 27–28, pp. 42–69, Aug. 2014.

[48] J. J. Carroll, "Signing RDF graphs," in *The Semantic Web* (Lecture Notes in Computer Science), vol. 2870, D. Fensel, K. P. Sycara, and J. Mylopoulos, Eds. Sanibel Island, FL, USA: Springer, 2003, pp. 369–384.
[49] C. Sayers and A. H. Karp, "Computing the digest of an RDF graph," Mobile Media Syst. Lab., HP Lab., Palo Alto, CA, USA, Tech. Rep. HPL-2003-235, 2004, vol. 1.
[50] M. Giereth, "On partial encryption of RDF-graphs," in *The Semantic Web* (Lecture Notes in Computer Science), vol. 3729. Galway, Ireland: Springer, 2005, pp. 308–322.
[51] C. Lantzaki, P. Papadakos, A. Analyti, and Y. Tzitzikas, "Radius-aware approximate blank node matching using signatures," *Knowl. Inf. Syst.*, vol. 50, no. 2, pp. 505–542, Feb. 2017.
[52] A. Hogan, "Canonical forms for isomorphic and equivalent RDF graphs: Algorithms for leaning and labelling blank nodes," *ACM Trans. Web*, vol. 11, no. 4, pp. 1–62, Jul. 2017, doi: 10.1145/3068333.
[53] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging," in *Proc. 18th USENIX Secur. Symp.*, Montreal, QC, Canada, F. Monrose, Ed., 2009, pp. 317–334.
[54] J. Mao, Y. Zhang, P. Li, T. Li, Q. Wu, and J. Liu:, "A position-aware Merkle tree for dynamic cloud data integrity verification," *Soft Comput.*, vol. 21, no. 8, pp. 2151–2164, 2017.
[55] Deadalnix's den. (2021). *Introducing Merklix Tree as an Unordered Merkle Tree on Steroid*. [Online]. Available: https://www.deadalnix.me/2016/09/24/introducing-merklix-tree-as-an-unordered-merkle-tree-on-steroid/
[56] A. Sutton and R. Samavi, "Integrity proofs for RDF graphs," *Open J. Semantic Web*, vol. 6, no. 1, pp. 1–18, 2019.
[57] R. Angles, H. Thakkar, and D. Tomaszuk, "RDF and property graphs interoperability: Status and issues," in *Proc. 13th Alberto Mendelzon Int. Workshop Found. Data Manage.*, vol. 2369. Asunción, Paraguay: CEUR-WS.org, 2019, pp. 1–11.

MIREK SOPEK received the M.Sc. degree in physics and the Ph.D. degree in computational chemistry from the Lodz University of Technology (TUL). He is currently an Entrepreneurial Scientist leading the Research and Development Team of MakoLab SA, a company he founded, in 1989, and helped expanding to international group of companies. He is the CEO of MakoLab USA Inc. He recently founded Quantum Blockchains, Inc., a startup devoted to protection of blockchains from the future risks.

DOMINIK TOMASZUK received the M.Sc. degree in computer science from the Bialystok University of Technology, Poland, in 2008, and the Ph.D. degree in computer science from the Warsaw University of Technology, Poland, in 2014. He is currently a Researcher at the Institute of Computer Science, University of Bialystok, Poland. His current research interests include semantic web, RDF, property graphs, NoSQL databases, and cheminformatics.

SZYMON GŁĄB received the M.Sc. degree in mathematics from the Lodz University of Technology (TUL), Poland, in 2002, and the Ph.D. degree in mathematics from the Polish Academy of Sciences, in 2007. He is currently a Researcher at the Institute of Mathematics, TUL. His current research interest includes pure mathematics.



FILIP TUROBOŚ received the M.Sc. degree in mathematics and the B.Sc. degree in computer science from the Lodz University of Technology (TUL), in 2017 and 2020, respectively. He is currently a Researcher at the Institute of Mathematics, TUL. His current research interests include some variants of the traveling salesperson problem and certain generalizations of metric spaces.

IVO ZIELIŃSKI received the B.Sc. degree in economics and business and the engineering degree in software engineering. He is currently a Blockchain Developer working as a Freelancer. He has participated in numerous research and development projects in the Fintech sector and his main specialization is Ethereum smart contracts.

DOMINIK KUZIŃSKI received the M.A. degree in economics. He has been an Employee of MakoLab SA, since 2006. Since 2012, he has been working at the Research and Development Department. He currently participates in projects related to blockchain technologies and graph databases.

RYSZARD OLEJNIK was a Researcher at the Lodz University of Technology (TUL), Poland. He is currently an Employee of the Research and Development Department, MakoLab SA. He deals with property graphs, NoSQL databases, blockchain, and programming. He participates in the works of the startup Quantum Blockchain Inc.

PIOTR ŁUNIEWSKI received the M.Sc. degree in applied mathematics from the Lodz University of Technology, in 1984. He is currently an Employee at the Research and Development Department, MakoLab SA. He participates in the works of the startup Quantum Blockchain. His research interests include property graphs, NoSQL databases, blockchain, and programming.

PRZEMYSŁAW GRĄDZKI is currently a Back-end Senior Developer and a System Administrator at MakoLab SA. He has hands-on experience in administering Linux systems, LAMP stack (and its variants), Tomcat/Jetty, PostgreSQL, and Docker. His research interests include the use of open-source products, especially related to Linux.