L2DART: A trust management system integrating blockchain and off-chain computation

ANDREA DE SALVE, Consiglio Nazionale delle Ricerche - ISASI, Italy
LUCA FRANCESCHI, University of Pisa, Italy
ANDREA LISI*, University of Pisa, Italy and Consiglio Nazionale delle Ricerche - IIT, Italy
PAOLO MORI, Consiglio Nazionale delle Ricerche - IIT, Italy
LAURA RICCI, University of Pisa, Italy

The blockchain technology has been gaining an increasing popularity in the last years, and smart contracts are being used for a growing number of applications in several scenarios. The execution of smart contracts on public blockchains can be invoked by any user with a transaction, although in many scenarios there would be the need for restricting the right of executing smart contracts only to a restricted set of users. To help deal with this issue, this paper proposes a system based on a popular access control framework called RT, Role-based Trust Management, to regulate smart contracts execution rights. The proposed system, called L2DART (Layer 2 DecentrAlized Role-based Trust management), implements the RT framework on a public blockchain, and it is designed as a layer-2 technology that involves both on-chain and off-chain functionalities to reduce the blockchain costs while keeping blockchain auditability, i.e., immutability and transparency. The on-chain costs of L2DART have been evaluated on Ethereum and compared with a previous solution implementing on-chain all the functionalities. The results show that the on-chain costs of L2DART are relatively low, making the system deployable in real-world scenarios.

CCS Concepts: • **Computing methodologies → Distributed computing methodologies**; • **Security and privacy → Distributed systems security**.

Additional Key Words and Phrases: Blockchain, Smart contract, Layer-2, Off-chain computation, Trust management, Access control

ACM Reference Format:
Andrea De Salve, Luca Franceschi, Andrea Lisi, Paolo Mori, and Laura Ricci. 2022. L2DART: A trust management system integrating blockchain and off-chain computation. *ACM Trans. Internet Technol.* 1, 1, Article 1 (January 2022), 29 pages. https://doi.org/10.1145/3561386

1 INTRODUCTION

Blockchain technology has been recently used as underlying infrastructure to implement a large number of distinct applications in several scenarios [3]. A blockchain is a distributed ledger shared among the members of a Peer To Peer (P2P) network that supports the execution of transactions that are meant to update the ledger status. Ethereum [52],

*Corresponding author: andrea.lisi@phd.unipi.it

Authors' addresses: Andrea De Salve, andrea.desalve@cnr.it, Consiglio Nazionale delle Ricerche - ISASI, Campus Universitario Ecotekne, Lecce, Italy, 73100; Luca Franceschi, University of Pisa, Largo Bruno Pontecorvo, 3, Pisa, Italy, 56127, l.franceschi5@studenti.unipi.it; Andrea Lisi, andrea.lisi@phd.unipi.it, University of Pisa, Largo Bruno Pontecorvo, 3, Pisa, Italy, 56127 and Consiglio Nazionale delle Ricerche - IIT, via G. Moruzzi, 1, Pisa, Italy, 56124; Paolo Mori, Consiglio Nazionale delle Ricerche - IIT, via G. Moruzzi, 1, Pisa, Italy, 56124, paolo.mori@iit.cnr.it; Laura Ricci, University of Pisa, Largo Bruno Pontecorvo, 3, Pisa, Italy, 56127, laura.ricci@unipi.it.

one of the most popular blockchains, supports the execution of Turing-complete programs known as *smart contracts* that implement decentralized applications. Smart contracts on public blockchains can be executed by anyone with a transaction. Therefore, similarly to many other Internet-based scenarios where the resources are shared among (potentially unknown) users, it may be required to restrict their usage to a set of trusted users that depends on a specific context. For this reason, a number of approaches for easily integrating access control functionalities in Ethereum smart contracts, such as OpenZeppelin [39] and [15], have been proposed in the literature.

The Role-Based Access Control (RBAC) model [24] is a well known and widely adopted method to regulate accesses to resources. In RBAC systems the owner of a *resource* defines a set of *roles* and associates the right to execute each *operation* on their resource to one (or even more) of these roles. When a user wants to execute an operation on a resource, the access decision process is performed to determine whether such user holds the role requested by the resource owner to perform such operation [46]. For instance, in a smart contract access control scenario, where the rights to execute smart contracts' functions must be regulated, the user deploying the smart contract (i.e., the resource owner) defines which role must be held to have the right to execute the functions a smart contract exposes. Then the resource owner associates to the other users one (or more) roles among the ones previously defined.

In order to infer if unknown users are trusted and eligible for a specific role, Trust Management Systems (TMSs) [4] have been introduced. To this aim, Li et al. [35] defined the Role-based Trust-management (RT) framework combining the strengths of RBAC and TMS. The RT framework allows its users to issue *trust credentials* that define, in terms of roles, the trust relations among them, as well as the rules to infer new trust relations from the existing ones. Hence, in the RT framework the roles of users are discovered at access request time using *search algorithms* [14], which exploit the trust credentials defined by all the users. The RT framework is suitable to improve the current state of the art of access control for smart contracts, especially in trans-organizational scenarios [13] where roles are assigned by multiple organization in collaboration with, or in behalf of, the system deployer.

In order to adopt an RT system to regulate the execution of smart contracts' functions, the RT system must be implemented on the blockchain as well. The advantages of building the RT system on top of a blockchain are several [2, 33, 53]. The blockchain takes care of both the storage and the processing of trust credentials, thus guaranteeing transactions immutability and transparency, as well as the correct evaluation of the trust credentials to infer new trust relations. Consequently, the blockchain-based RT framework benefits from data and computational *auditability*, i.e., anyone at any moment can read the available trust credentials, and can check the results obtained from their processing. Auditability is a relevant feature [33] because no party should be able to misbehave, e.g., assigning or revoking a role, without the others knowing that, and no party can repudiate the actions they performed [13, 15]. Moreover, performing the role inference process on the blockchain prevents a potential malicious resource owner to state false claims, e.g., denying a requested access even though the requesting user holds the specified role.

In this respect, in a previous work [25] we focused on *public and permissionless* blockchains presenting DART, an Ethereum implementation of a subset of RT called RT_0. To the best of our knowledge, the RT framework is not supported by any other existing access control systems for smart contracts.

In DART, the trust credentials defined by users are stored and evaluated on the blockchain. The DART smart contract allows its users to create such credentials implementing RT_0, and exposes an algorithm, called *backward search* algorithm, which infers the users having a specific role from the existing trust credentials. Consequently, the blockchain guarantees the immutability and a correct processing of the trust credentials without the need of trusted intermediaries.

1.1 Motivation and contributions

The implementation of a decentralized RT framework fully based on a public blockchain can prove to be a challenge because smart contracts cannot currently perform complex computations on-chain. Indeed, from the experiments we conducted in [25] on an access control scenario implemented on Ethereum, we found out that the gas consumed by the DART smart contract to find the users holding a given role is very large. In particular, DART overcomes the Ethereum block gas limit when processing the trust credentials of 20 users belonging to more than 15 organizations (universities in our experiment). This could prevent the deployment of DART in several real use cases, where considerably larger problems must be taken into account.

To overcome the scalability problem of DART while keeping the auditability and decentralization of public blockchains, in this paper we enhanced DART making it a layer-2 system following the off-chain computation model [16], in particular applying the verifiable computation approach [17]. The new framework, named L2DART (Layer-2 DecentrAlized Role-based Trust management), is based on the intuition that in this scenario computing a solution off-chain and verifying it on the blockchain is considerably cheaper than computing such solution on the blockchain.

L2DART stores RT_0 credentials on a public and permissionless blockchain in the same way as DART. Instead, for what concerns the inference of users' roles from existing trust credentials, L2DART requests its users to run the backward search algorithm off-chain, i.e., on their premises, exploiting the credential available on the blockchain. Together with the result, the algorithm produces a proof validating it. This proof will be evaluated on the blockchain, by the L2DART smart contract, in order to verify that the correspondent result is correct, i.e., a user holds a specific role according to the existing trust credentials. This is particularly useful in trans-organizational Role-based access control systems [13], where roles can be assigned to principals also by other organizations than the one that deployed the smart contract, and the role required to execute the smart contract can then be inferred composing such roles.

Based on the motivations explained above, this paper provides the following contributions:

- The design of a layer-2 system, L2DART, a Role Based TMS implemented on top of a public and permissionless blockchain that allows to regulate smart contracts' execution rights in dynamic and trans-organizational scenarios. L2DART makes TMSs benefiting from blockchain auditability while keeping their execution costs on the blockchain affordable;
- The implementation of a prototype of the proposed system, following the latest state of the art best practices, consisting of an on-chain module as a Solidity smart contract and an off-chain module as a Python software;
- A quantitative evaluation of the costs of L2DART in three application scenarios, a comparison with the costs of DART, and a qualitative discussion.

The rest of the paper is organized as follows. Section 2 presents the fundamental concepts related to the blockchain layer-2 technologies and Trust Management Systems. Section 3 presents L2DART and the problem it tackles, where a new verifiable computation protocol is introduced, while Section 4 describes the approach in detail. Section 5 presents the implementation of L2DART with Ethereum smart contract and a Python module, it shows the costs focusing on the gas metrics of Ethereum, and it compares such costs with a prototype presented in a previous work. Finally, Section 6 discusses the system, Section 7 compares it with the related work on access control systems implemented on blockchain, and Section 8 outlines the final remarks and future work.

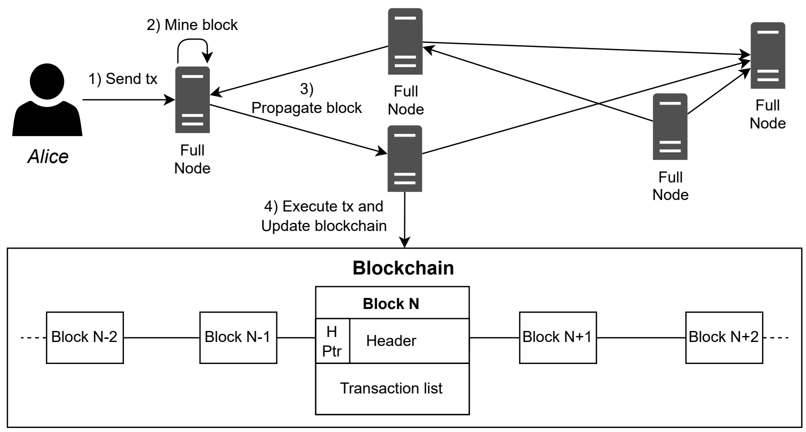


Fig. 1. Overview of a blockchain network.

2 BACKGROUND

2.1 Blockchain and off-chain computation

As shown in Figure 1, a blockchain is an append-only list of blocks, each composed of a header and a list of transactions, which are cryptographically linked by a hash pointer stored in the header (denoted as H Ptr in Figure 1). A Blockchain is typically managed by the nodes of a peer-to-peer network that execute a consensus protocol to achieve agreement on the next block to be added. A user Alice sends, by means of a wallet application, a transaction, Tx, to the peer-to-peer network. The transaction Tx is stored in the transaction list of a new block created (mined) by one of the nodes. Such block is propagated, each node re-executes the transactions in the transaction list and stores the new block in its local copy of the blockchain. While sending her transaction, Alice needs to pay a fee for the execution and storage of the transaction.

According to research [10, 32, 56], scalability and transaction cost are the two main problems that hinder a wide usage of blockchain technology. For instance, the Bitcoin blockchain can process on average 4 transactions per seconds (TPS) with an average transaction fee equal to 183.61 USD on October 15th 2021 [54]. Instead, the Ethereum blockchain executes about 14 TPS and the average cost of a single transition is equal to 11.38 USD on October 16th 2021 [55]. Such limitations led to a severe network congestion of the Ethereum blockchain in 2018, when the CryptoKitties Decentralized application became popular among users.

To tackle this issue, *layer-2* models [31, 56] have been proposed. These models build an overlay connected to the blockchain able to perform operations that execute independently of the consensus protocol, but bound to the blockchain with specific on-chain transactions [28]. The goal of layer-2 models is to reduce the code that on-chain transactions execute making them responsible of connecting an off-chain operation with the blockchain [28]. As a consequence, the transaction cost is smaller, more transactions can be placed in each block and, consequently, the time a transaction has to wait before being placed in a block could be shorter. The advantages are, therefore, reduced transactions costs and latency, increased transaction throughput, but also increased privacy since not all the transactions are executed on-chain. Similarly, other layer-2 models have been designed and developed. For instance, the Bitcoin Lightning [43] and the Ethereum Raiden Networks [44] implement an *off-chain channels* model, i.e., they create virtual channels that allow two users to exchange cryptocurrency independently from the blockchain consensus, and exploit the network of

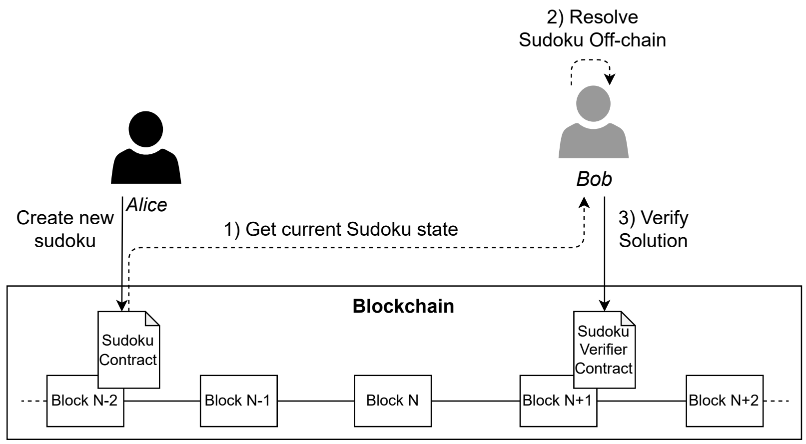


Fig. 2. Off-chain operations on smart contracts.

off-chain channels to route payments among users that are not directly connected by a channel. The *side-chain* model, instead, is designed to connect parallel chains either with an existing blockchain, for example in the Plasma project for Ethereum [42] (now deprecated [21]), or with a brand new blockchain, such as on Cosmos [34] and Polkadot [51]. This model "splits" the blockchain in several side-chains, each side-chain processing transactions in parallel with the others, and a mainchain that verifies the correctness of the sidechain operations. Finally, a similar model is known as *cross-chain*, which connects existing blockchains, for example with cross-chain atomic swaps [29] or bridging approaches [47].

In this paper, we focus on the *off-chain computation* model, where an intensive computational task is outsourced to nodes external the blockchain, while the blockchain stores application's data that will be used in the future to verify the correctness of the off-chain result. Additionally, a verification algorithm can be implemented on-chain, which must be cheap, to validate the off-chain result with the goal of guaranteeing blockchain auditability. Following this protocol, known as *verifiable computation* [16], a Prover executes a computation producing a result along with a proof attesting the computation's correctness, and publishes the proof on the blockchain. A Verifier verifies the proof and confirms the result if the proof is correct. This protocol should be non interactive, i.e., the protocol must make use of a single message, the verification must be cheap, the security assumptions on the Verifier must be weak to not introduce additional trust in conflict with the blockchain's purpose, and zero-knowledge properties could be integrated if private inputs are required. For example, a user Alice could publish a sudoku on the blockchain, and another user Bob could solve such sudoku off-chain and publish on the blockchain the solution or a proof of it, whose correctness is easy to verify. Figure 2 shows a blockchain with two smart contracts, one to create a new sudoku game and the other that verifies if a sudoku has been solved correctly. Alice invokes the first smart contract to create a new sudoku, while Bob reads the current sudoku from the blockchain, solves it off-chain, and invokes the second smart contract to verify the correctness of his solution and to store it on-chain.

As a result of their research in off-chain computation, Eberhardt et al. [17] proposed a list of off-chaining patterns. We describe those applied in this paper: in the *challenge and response* pattern, a smart contract only accepts state transitions, challenges, and a confirmation, or a rejection, of the challenges; in the *delegated computation* pattern, a user outsources a heavy computation to an off-chain node, which provides both the result and a proof of correctness that can be verified on-chain. Other patterns are the off-chain signatures, the content-addressable storage, and the delegated computation patterns. Given the transparent nature of most blockchains, the verification process publicly exposes the data because it needs a transaction. To mitigate this issue, researchers studied the verification of zero-knowledge proofs on smart contracts [38, 41] to prove that users have a certain property, such as the age or the salary, above, below, or within a range of valid values without revealing the value itself. A notable tool that integrates zero-knowledge proofs with Ethereum is ZoKrates [18]. Finally, other off-chain computation tools are Truebit and ARPA. Truebit [49] is an off-chain verification tool that aims to overcome the computational limitations of a decentralized network. In Truebit, a *Taskgiver* requests a computational heavy task storing an entry in a smart contract, *Solvers* offer themselves to solve it in exchange of a reward, and *Verifiers* check the correctness of the result. ARPA [5], instead, is a Multiparty Computation (MPC) network. In an MPC network a set of n parties, including an adversary, wish to learn the outcome of a function $y = f(x_1, x_2, ..., x_n)$ where a participant i knows only their secret input x_i, and the outcome y must be correct [12]. The ARPA network is composed by a set of nodes performing secured and privacy preserving computation in a MPC fashion, communicating with a blockchain through a proxy smart contract that stores ARPA computation requests. The goal is to provide a verifiable scheme to prove that a certain computation has been performed off-chain by the ARPA network, while protecting the privacy of the content and of the participants.

2.2 Role-Based Trust Management Systems

A TMS is defined as a set of principles used to model collaborative authorization modules capable of managing the access over shared resources [8]. The Role-Based Trust Management system framework RT [35, 36] brings together RBAC and TMSs to regulate the access to resources in an environment involving multiple independent organizations. As shown in Figure 3, the key elements of RT are: *i) principals* (or entities), i.e., the users of the system who can issue trust credentials or request for an authorization; *ii) trust credentials*, i.e., rules describing trust relationships between principals through roles; *iii) policies*, i.e., sets of trust credentials representing the rules to evaluate in order to authorize a request.

2.2.1 Principals, roles, and credentials. In RT, a *principal* reflects a user, and a *role* is created by a principal responsible to define appropriate trust relationships on that role for a specific domain of interest. A role is denoted by the name of the principal who defined it followed by a *role name* and separated by a dot. A principal cannot modify the roles created by other principals, but the principal can use them to extend their trust relationships. For example, only the principal *University* can create the roles *University.student* and *University.professor*, where *student* and *professor* are the role names, which define the student and professor roles within the organization. After a set of roles has been defined, the principal who created them can assign such roles to other principals with credentials.

A *credential* defines the rule to assign a principal to a defined role or, in other words, it states whether a principal is a member of a role. For example, if *University* associates the role *University.student* to *Bob*, we say that *Bob* is a member of the role *University.student.* A principal can assign *members*, directly or via delegation, only to their roles. For example, only *University* can directly assign *Bob* as a member of *University.student*, or can delegate the assignment to another principal. The set of credentials forms a *policy* and it can be assumed that principal names are unique in a policy [36].

Let *Alice*, *Bob*, and *Charlie* be principals and r, s, t be role names. The RT^0 language, a subset of RT, defines the following types of credentials [36] in the form of "*assigned-role* ← *role-expression*":

- Simple member: $Alice.r \leftarrow Bob$
  Alice asserts that Bob is assigned the role $Alice.r$;

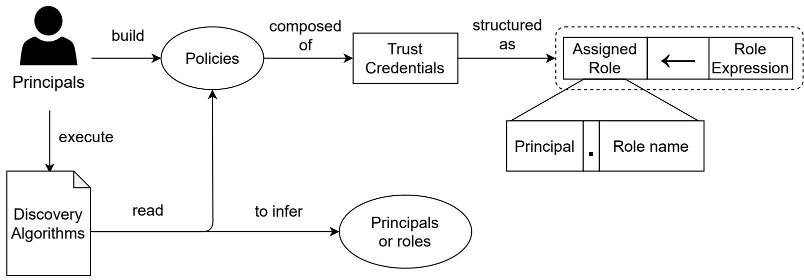


Fig. 3. Workflow of the RT framework.

- Simple inclusion: $Alice.r \longleftarrow Bob.s$
  Alice delegates to Bob the assignment of the role $Alice.r$ on her behalf. In the example, all the members of the role $Bob.s$ are also considered to be members of the role $Alice.r$;
- Linked inclusion: $Alice.r \longleftarrow Bob.s.t$
  Alice uses the linked inclusion to include into $Alice.r$ all the members of $P.t$, for every principal P that is member of $Bob.s$. In the example, $Alice.r \longleftarrow P.t\ \forall P \in Bob.s$;
- Intersection inclusion: $Alice.r \longleftarrow Bob.s \cap Charlie.t$
  This credential assigns the role $Alice.r$ to all the principals having both the roles $Bob.s$ and $Charlie.t$.

Unless stated explicitly, capital letters such as A, B, P, represent principals and $A.r$, $A.s$, etc represent roles.

2.2.2 Weighted credentials. The credentials provided by RT^0 are defined according to an "hard" security approach [1], where a credential either assigns a role to a principal or not. Approaches extending RT^0 with weights have been presented in [6] where the authors define RT^W to assign a weight to the simple member credentials, whose semantic depends on a certain c-semiring tuple (maximize a value or minimize a cost), and in [22] where the authors build a reputation and a recommendation model defining two families of credentials.

2.2.3 Backward search chain discovery algorithm. The *backward search chain discovery algorithm* [14] is a discovery algorithm that is invoked by a principal, as shown in Figure 3, and answers to the following query: given a policy $\mathcal{P}$, find the set of principals $\{pi\}$ that hold (are member of) an input role $A.r$.

The algorithm navigates the trust credentials in $\mathcal{P}$ to build a *proof graph*, i.e., a data structure that represents through the *nodes* the role-expressions and the assigned-roles present in $\mathcal{P}$, and through *edges* the relationships among such nodes according to the trust credentials. Each node also stores the list of principals, also known as *solutions*, that are found by the algorithm to be members of the role-expression or the assigned-role represented by that node. Each time a solution is added to the solution set of a node n and n has an outgoing edge to n', the solution is also added, we say *propagated*, to the solution set of n'. An example of proof graph is shown in Figure 4 where, for simplicity, the solutions stored in the nodes are not shown. The algorithm begins initializing a queue of nodes and the proof graph both with a single node representing the input role $A.r$. As long as the queue is not empty, the algorithm removes the first node n from the queue and process it as follows:

(1) when a node n representing a role $A.s$ is processed, the algorithm finds the trust credentials in $\mathcal{P}$ having $A.s$ as assigned-role (left-hand side of a credential) and, for each role-expression (right-hand side of a credential) it creates the corresponding node, n', it adds n' to proof graph, it creates an edge from n' to n, and it adds n' to

the queue. The pair (n, n') and the edge connecting them correspond to a credential in $\mathcal{P}$.
Note that the new node n' is added to the proof graph and to the queue only if a node representing the same role-expression does not already exist in the proof graph. This check and the related actions are performed also during the following steps, but we omit it to simplify the descriptions. If n' already exists, then a new edge is created from n' to n. If n' also contains solutions, then such solutions are propagated from n' to n using the new edge. How a solution is added for the first time to the proof graph is explained in step 4;

(2) when a node n representing a linked inclusion $A.s.t$ is processed, the algorithm creates a node n' for $A.s$ and adds n' in the proof graph and to the queue. The solutions of $A.s.t$ are all the principals $pi \in B.t : \forall B \in A.s$. To find solutions of $A.s.t$, the algorithm instantiates a "monitor" object, called *L_Monitor*, on the node n which observes the solution set of the node n' ($A.s$): each time a new solution B is added to such solution set, the monitor creates a node n'' in the proof graph representing $B.t$, adds n'' to the queue, and creates an edge from n'' ($B.t$) to n ($A.s.t$). This edge is known as *derived edge* because there is no credential in $\mathcal{P}$ directly representing it, i.e., in the form $A.s.t \leftarrow B.t$, but it is derived by the combination of semantically equivalent credentials that proved B to be a solution of $A.s$: these credentials are known to be the *support set* of the derived edge;

(3) when a node n representing an intersection inclusion $B.s \cap C.t$ is processed, the algorithm creates a node for each role of the intersection, i.e., it creates a node n' for $B.s$ and a node n'' for $C.t$, and it adds them to the proof graph and to the queue. Similarly to the linked inclusion node, to understand if a principal pi is member of both roles, the algorithm instantiates a "monitor" object, called *I_Monitor*, on the node n of the proof graph which observes the solution set of both the nodes n' ($B.s$) and n'' ($C.t$). Each time a principal pi is added to the solution set of n' (or n''), the monitor checks if pi is also present in the solution set of n'' (or n'): if the answer is positive, we say that the monitor is activated by pi;

(4) when a node n representing a principal pi is processed, the algorithm begins the propagation of pi through the proof graph. The propagation of a solution pi through the proof graph is performed by recursively performing the propagation step on the node n as follows. The algorithm examines all the nodes n^j reachable from n following its outgoing edges, including the derived ones. For each of these nodes n^j, the propagation step is recursively performed on n^j if and only if pi is not already present in the solution set of n^j, and pi is added to the solution set of n^j. Moreover, if n is monitored by a I_Monitor attached to an intersection node n^k and pi activates such monitor (step 3), the algorithm adds pi to the solution set of n^k and recursively executes the propagation step on such node as well. Recall that the propagation of solutions also happens when a new edge, also a derived one, e is created among existing nodes of the proof graph, say from n to n', to propagate the solutions already stored in n to the nodes of the proof graph that are now reachable through e (as anticipated in step 1).

By construction, the graph ensures that if the principal pi has role $A.r$, then there exists a path from the node representing pi to the node representing $A.r$ (completeness). Furthermore, if a path from node $A.r$ to the node of the principal pi exists, then the principal pi has role $A.r$ (soundness) [35]. Once the queue is empty, meaning the algorithm terminated, the set of solutions stored in the node representing the input role $A.r$ is returned as the answer to the initial query. Therefore, the backward search chain discovery algorithm infers the set of principals having the role $A.r$ by properly combining the existing credentials. Instead, the *forward search chain discovery algorithm*, see [14] as well, infers from the credentials in $\mathcal{P}$ the set of roles held by a given principal pi.

$$EPapers.studentMember \leftarrow EOrg.member \cap EOrg.student \qquad (1)$$

$$EOrg.student \leftarrow EOrg.university.student \qquad (2)$$

$$\begin{array}{llll}
(3.1)\ EOrg.university \leftarrow StateA.university & (3.2)\ EOrg.university \leftarrow StateB.university & (3.3)\ [...] & (3) \\
(4.1)\ StateA.university \leftarrow UniA1 & (4.2)\ StateA.university \leftarrow UniA2 & (4.3)\ [...] & (4) \\
(5.1)\ StateB.university \leftarrow UniB1 & (5.2)\ StateB.university \leftarrow UniB2 & (5.3)\ [...] & (5) \\
(6.1)\ UniA1.student \leftarrow Alice & (6.2)\ UniA1.student \leftarrow Bob & (6.3)\ [...] & (6) \\
(7.1)\ UniB1.student \leftarrow Charlie & (7.2)\ UniB1.student \leftarrow Dave & (7.3)\ [...] & (5)
\end{array}$$

$$EOrg.member \leftarrow Alice \qquad (8)$$

2.2.4 Sample policy. This section shows the application of the backward search algorithm to a sample policy, derived from [14] and named $\mathcal{P}_{Epapers}$ that will be used as reference example through the paper. For instance, the policy could be used to grant the right to the members of the role *EPapers.studentMember* to access some educational material with a discount. In the following representation of the policy $\mathcal{P}_{Epapers}$, similar credentials have been placed on the same line.

Figure 4 shows the proof graph built as a result of the execution of the backward search chain discovery algorithm with $\mathcal{P}_{Epapers}$ as input policy and $EPapers.studentMember$ as input role, which is the entry point of the graph.

Each box represents a role-expression or an assigned-role, the plain arrows connect pairs of nodes in order to represent the credentials in $\mathcal{P}$. The dashed arrows represents the derived edges built by L_Monitor: for example, the derived edge (a) is created as a result of edges (3.1) and (4.1), which are the support set of (a). The monitors L_Monitor and I_Monitor are represented as blue boxes right below the role-expression they support, and the dotted blue lines connect the monitors to the nodes the monitors observe. Finally, Figure 4 highlights in red the minimal subset of the nodes and edges of the graph to find that Alice is a member of $EPapers.studentMember$.

3 A LAYER-2 DECENTRALIZED ROLE-BASED TRUST MANAGEMENT

This section describes the design of **L2DART**, a Role-Based Trust Management System implementing the RT^0 framework as a blockchain layer-2 system that is used to regulate the execution of smart contracts' functions. L2DART is an enhancement of a previous system, DART [25], which implements the RT^0 framework entirely on the blockchain, but it overcomes the Ethereum gas limit already when the number of trust credentials in a policy is relatively low, thus being not usable in real scenarios. The main idea underlying the L2DART approach is to outsource the execution of the search algorithm, which is computationally intensive, to a service running outside the blockchain, still maintaining the advantages of the blockchain by introducing a result verification step executed on the blockchain.

In our reference application, L2DART is used to protect smart contracts, i.e., to regulate the rights of blockchain users to execute the functions exposed by such contracts. To this aim, the smart contract developers decide which role must be held by users to execute each function exposed by their smart contracts, while the protected smart contract, before executing their functions, must invoke the L2DART smart contract to check that the caller actually holds the required role. Furthermore, a constraint on the minimum weight paired to the required role can be imposed as well. The integration of the invocation to L2DART in the protected smart contracts is very simple, and could be executed even by an automatic inlining tool. As a matter of fact, it is sufficient to add a call to the L2DART smart contract among the first instructions of the code of each function. For example, in Solidity this could be implemented with a modifier.

In the following of this section, Section 3.1 summarizes DART, the groundwork of the proposed approach, while Section 3.2 describes L2DART, an improvement of DART exploiting the layer-2 blockchain technology to solve the DART issues while maintaining the properties of blockchains.

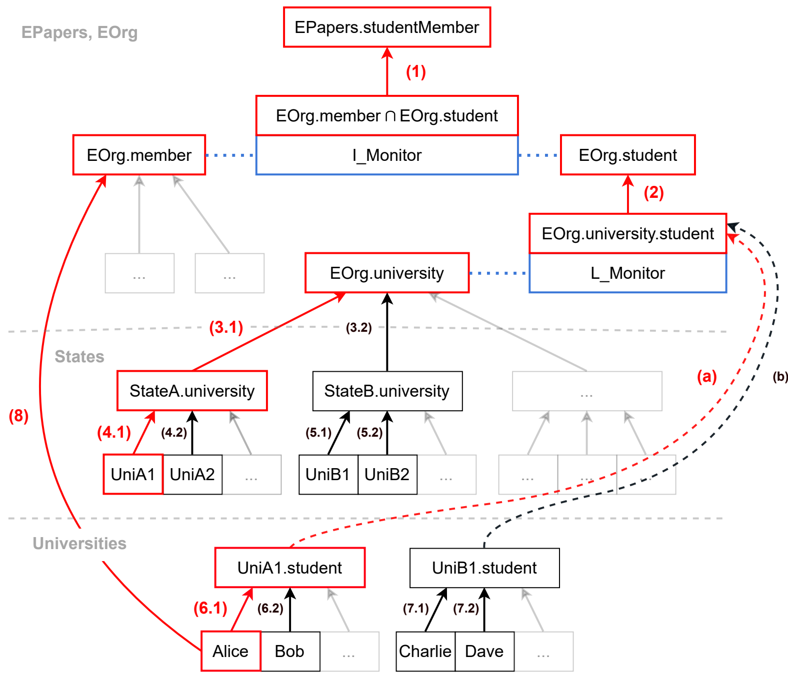


Fig. 4. Illustration of the proof graph generated by the execution of the backward search chain discovery algorithm to find the users having the role *EPapers.studentMember* according to the policy $\mathcal{P}_{Epapers}$. Highlighted in red the subset of nodes and edges related to the solution Alice.

3.1 DART

DART has been presented in [25], and it is a Role-Based Trust Management System implementing the RT^0 framework (see Section 2.2) on a public blockchain. The DART smart contract allows its users to create new trust credentials and to execute the backward search discovery algorithm (described in Section 2.2.3) to find the users holding a given role $A.r$ according to the policy. DART supports weights attached to the trust credentials similarly to the solutions listed in Section 2.2.2. Since DART is implemented on a public blockchain, it inherits the blockchain advantages: the trust credentials building the policy are public and robust against modification and censorship, and any user can independently process a policy by executing the DART chain discovery algorithm on the blockchain. The code implementing DART is publicly available [26].

However, the properties listed above come with a cost. Executing transactions on public blockchains require a fee to be paid in cryptocurrency, which might be large when the computation is complex. Indeed, the experimental evaluation we conducted on the Ethereum implementation of DART (described in [25]) showed us that, while storing credentials has reasonable costs, executing the backward search algorithm on the blockchain has an high cost. Such cost overcomes the block gas limit when the complexity of the policy increases, making DART not usable in many real scenarios.

3.2 The design of a Layer-2 DART

In order to overcome the previously described limitations, we enhanced DART by redesigning it as a layer-2 system, L2DART, in order to satisfy the following properties:

- P1 **Data auditability**: the trust credentials are permanently stored on the blockchain and benefit of blockchain auditability, i.e., anyone can independently read these credentials at any time;
- P2 **Computational auditability**: the verification of a role must benefit of blockchain auditability, i.e., anyone can reliably verify that a user really held a given role at a given time according to the trust credentials present at that time;
- P3 **Affordable fees**: the system should be usable for real applications, i.e., the fees to pay to store trust credentials and to process them must be affordable.

In the following, we take as example the policy $\mathcal{P}_{Epapers}$, presented in Section 2.2.4, and we assume Alice wants to prove to EPapers that she is a member of $EPapers.studentMember$.

DART satisfies the property P1 because the DART smart contract stores all the trust credentials composing $\mathcal{P}_{Epapers}$. Theoretically, DART also satisfies the property P2 because it allows anyone to prove that Alice holds the role $EPapers.studentMember$ since the backward search algorithm is implemented by the DART smart contract as well. However, this computation requires high fees overcoming the Ethereum block gas limit even with simple policies (see the experiments in Section 5.1, Test Scenario A), thus not satisfying the property P3, nor P2 in practice. Moreover, the backward search algorithm finds all the users holding the role $EPapers.studentMember$, information that Alice does not need to compute, and therefore to pay for being computed on the blockchain. Alternatively, the forward search algorithm, which computes all the roles of a principal [14], could be used, but it has the same issues of the backward search algorithm, since *i)* it computes more solutions than the required one, and *ii)* it needs high fees to search for all the solutions on-chain. In both cases, the extra solutions returned by the algorithms could not be cached to avoid future invocation of the algorithms. As a matter of fact, policies are dynamic because existing credentials could be removed, while new credentials could be added. Consequently a solution computed at time T_1 may not be valid any more at time T_2, where $T_2 > T_1$, because the set of valid credentials in the policy could be changed. In this respect, the choice of the most suitable algorithm depends on the specific problem to be addressed. For instance, in a policy describing a web of trust scenario (see the experiments in Section 5.1, Scenario B), a principal Eve trusts another principal if the latter holds the role $Eve.trust$. In this scenario, Eve might want to know all the principals she trusts, i.e., all the principals having the role $Eve.trust$ in the policy. In this case, the backward search algorithm provides the answer to this query. Alternatively, in the same scenario, a principal, say Alice, who wants to know all the other principals pi who trust her, needs to discover all the roles $pi.trust$ she holds according to the policy. In this case, the forward search algorithm provides Alice with the answer to this query. Instead, in the access control scenario a principal must have a given role to be authorized to invoke a given function of a given smart contract. In this case, both the backward search algorithm and the forward search algorithm are suitable because the solutions they compute include the required one. In this paper we focus the attention on the backward search algorithm, but the approach can be extended to the forward search algorithm as well. Indeed, to meet the properties P2 and P3, L2DART executes the backward search algorithm off-chain following the off-chain computation model. As a matter of fact, L2DART implements the verifiable computation protocol and uses the blockchain to verify, among the solutions computed off-chain, only the ones needed to guarantee property P2. This significantly reduces the cost of the code executed on the blockchain and also prevents L2DART users to be charged by the blockchain for the computation of the solutions they do not need, thus satisfying also property P3.

We designed the L2DART architecture and operations as follows:

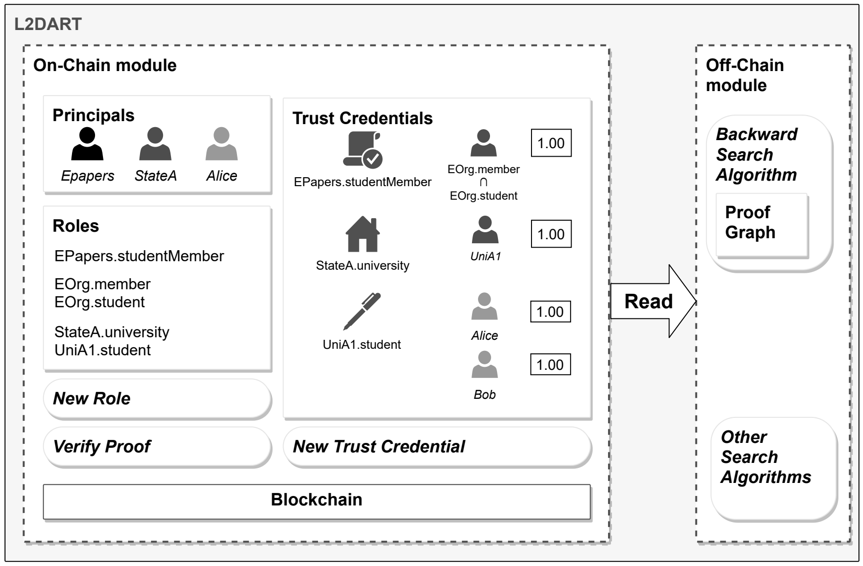


Fig. 5. A representation of the L2DART architecture.

(1) L2DART is composed by two modules: an on-chain module, i.e., the L2DART smart contract deployed on a public blockchain, and an off-chain module, i.e., the L2DART application deployed on one or more computers external the blockchain;
(2) The on-chain module allows users to upload their RT^0 trust credentials enhanced with a trust weight value on the blockchain. This is unchanged from DART;
(3) The off-chain module exposes a function implementing the backward search algorithm, called *OFFchainBackwardSearch*. Given in input a policy and a role, the *OFFchainBackwardSearch* function returns a, possibly empty, list of triples *(principal, weight, proof)* indicating that *principal* is member of the input role, within the input policy, with a certain *weight* according to a *proof* which demonstrates it;
(4) When a user wants to prove they hold a role to obtain the right to execute a L2DART protected smart contract *SC*, they need to provide to *SC* the proof produced by the off-chain module. In turn, *SC* calls the *verify* function exposed by the on-chain module to check the validity of such proof: the module processes the proof and returns, as a result, the role the proof demonstrates, or an error if the proof is not well formatted. Depending on the result of the proof, *SC* will grant or deny the execution right to the user.

Figure 5 shows an overview of the L2DART architecture, consisting of the two modules. As depicted, the on-chain module provides the functionalities to store the principals, the roles, and the trust credentials of a policy on the blockchain, and exposes the *verify* function and the functions to store roles and credentials. The off-chain module allows its users to execute the chain discovery algorithm on their computers, using the trust credentials read from the on-chain module. We assume the off-chain module to be untrusted, i.e., it may craft malicious results, and to not have any computational limitations (i.e., they can execute the search algorithms even for very complex policies). Instead, since the on-chain module is executed by the blockchain nodes, it is trusted, but it has a limited computational capacity and a fee must be paid for its execution. These assumptions will be discussed in Section 6.

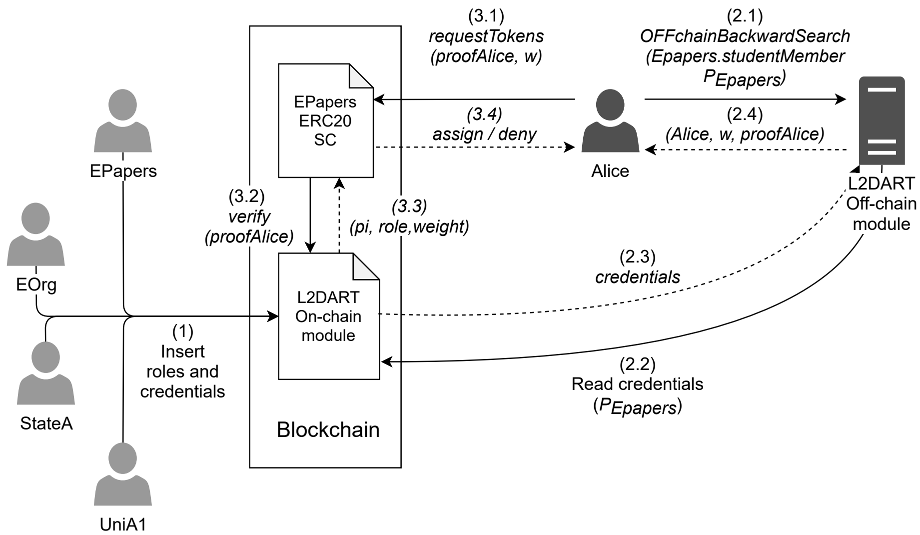


Fig. 6. The typical sequence of operations when using L2DART.

Figure 6 shows the sequence of the L2DART operations applied to the policy $\mathcal{P}_{Epapers}$. Plain arrows represent function calls, while dashed arrows represent the related answers. The principals $EPapers$, $EOrg$, $StateA$, and $UniA1$ cooperatively build the policy by invoking the L2DART on-chain module to create the roles and the credentials listed in Section 2.2.4 (step 1). We assume that *EPapers* deployed a smart contract called *EPapers ERC20 SC* that assigns ERC20 tokens [11] to the users invoking it having the role $EPapers.studentMember$, for example as a coupon to spend on educational material. Alice wants to execute the EPapers ERC20 SC smart contract in order to obtain such tokens, therefore she needs to prove she has the required role. Using L2DART, Alice queries the off-chain module, executed on her personal computer, to execute the *OFFchainBackwardSearch* function to produce the proof to be shown to *EPapers ERC20 SC* smart contract (step 2.1). The off-chain module reads the credentials from the on-chain module (steps 2.2 and 2.3), and it executes the algorithm to return to Alice the solution about her that includes her principal name $Alice$, the trust value w, and the proof $proofAlice$ (step 2.4). Afterwards, Alice provides the proof and the trust value to the *EPapers ERC20 SC* smart contract (step 3.1) that, in turn, invokes the L2DART on-chain module to execute the proof verification (step 3.2). If the verification does not raise an error, the on-chain module returns a tuple consisting of a principal pi, a $role$, and trust value $weight$ (step 3.3). Finally, the smart contract *EPapers ERC20 SC* checks that pi is equal to $Alice$ and that $role$ is equal to $EPapers.studentMember$[1]: if such checks are passed, the smart contract is executed, thus assigning to Alice the tokens, otherwise the execution is denied and no token is assigned (step 3.4).

As ensured by P2, computational auditability, *Alice* generates a valid proof (by using the *OFFchainBackwardSearch* algorithm), which will be verified on the blockchain without the need to trust *EPapers*.

4 SYSTEM ARCHITECTURE AND APPROACH

This section describes in details the three phases of the L2DART system workflow, as shown in Figure 6. Section 4.1 describes the first phase of the workflow, the creation a L2DART policy. Section 4.2 describes the second phase, the

[1]In this policy all weights are equal to 1, therefore we can assume this is not a parameter to check.

execution of *OFFchainBackwardSearch* on the off-chain module to generate a result and a proof. Section 4.3 describes the third phase, the verification of a proof on the blockchain to confirm that a principal is actually member of a given role.

4.1 Phase 1: Build a policy

The on-chain module stores the trust credentials composing the policy. In particular, each principal A is enabled to create only their roles (e.g., $A.r$, $A.s$, etc.), and contributes to build the policy by uploading the RT^0 credentials, which represent their trust relations. For example, several principals participated to build the policy $\mathcal{P}_{EPapers}$: the principal $UniA1$ creates the role $UniA1.student$ and the credential $UniA1.student \longleftarrow Alice$; the principal $EPapers$ creates the role $EPapers.studentMember$ and the credential $EPapers.studentMember \longleftarrow EOrg.member \cap EOrg.student$, where $EOrg.member$ and $EOrg.student$ are two roles previously created by the principal $EOrg$ in the same policy. Storage details are described in [25].

4.2 Phase 2: Compute role members and generate the proofs

The off-chain module exposes the *OFFchainBackwardSearch* function, which takes as input a role name, e.g., $A.r$, and a policy (represented by its address on the blockchain), it reads the trust credentials representing the policy from the blockchain querying the on-chain module, and it returns a list of solutions each represented by a triple (pi, w_{pi}, τ_{pi}) meaning that the principal pi is a member of the input role $A.r$ with weight w_{pi} and τ_{pi} is the proof demonstrating it.

In particular, the proof τ_{pi} is the list of trust credentials that are paired with the arcs of the proof graph that have been taken into account by the *OFFchainBackwardSearch* algorithm to determine that the principal pi holds the input role $A.r$. In case there are two, or more, paths in the proof graph proving the same solution (i.e., assigning to a principal the same role), the algorithm keeps the path that gives to the principal the highest weight.

In the following of this section, we describe how τ_{pi} is constructed from the policy $\mathcal{P}$ while executing the *OFFchain-BackwardSearch(A.r, $\mathcal{P}$)* function. For simplicity, in the following we use the symbol $\mathcal{P}$ to represent both a policy and the address of the smart contract that stores it on the blockchain.

We refer to the steps used in the description of the backward search algorithm in Section 2.2.3. When a new principal pi is added to the proof graph (step 4 of the algorithm), the solution is represented by a tuple $(pi, \_, \{\})$ (having an empty proof) paired to node n. Each time the solution is propagated through an edge, the solution is updated as follows:

- The first edge that is followed by the algorithm to propagate the solution is always an edge created by a simple member credential of the form $A.s \longleftarrow^{w} pi$: in this case, the initially empty solution is updated to $(pi, w, \{A.s \longleftarrow^{w} pi\})$;
- Each time the algorithm propagates the solution by following an edge generated by a credential $A.s \longleftarrow^{w} roleExpr$, the solution (pi, w_{pi}, τ_{pi}) is updated to $(pi, w_{pi} * w, \tau_{pi}.append(A.s \longleftarrow^{w} roleExpr))$, where *roleExpr* can either be a role $B.t$, a linked role $A.s.t$, or an intersection role $B.s \cap C.t$;
- When a I_monitor is activated on an intersection node $B.s \cap C.t$, the node receives two solutions: $(pi, w_{pi}^{B.s}, \tau_{pi}^{B.s})$ from the node $B.s$, and $(pi, w_{pi}^{C.t}, \tau_{pi}^{C.t})$ from the node $C.t$. The resulting solution is $(pi, min(w_{pi}^{B.s}, w_{pi}^{C.t}), \tau_{\cap})$ where:
$$\tau_{\cap} = \begin{cases} \tau_{pi}^{B.s}.concat(\tau_{pi}^{C.t}) & \text{if } len(\tau_{pi}^{B.s}) > len(\tau_{pi}^{C.t}) \\ \tau_{pi}^{C.t}.concat(\tau_{pi}^{B.s}) & \text{otherwise} \end{cases}$$
i.e., the resulting solution takes the minimum weight received, and its resulting proof is composed by the longest proof between $\tau_{pi}^{B.s}$ and $\tau_{pi}^{C.t}$ attached before the shortest one;

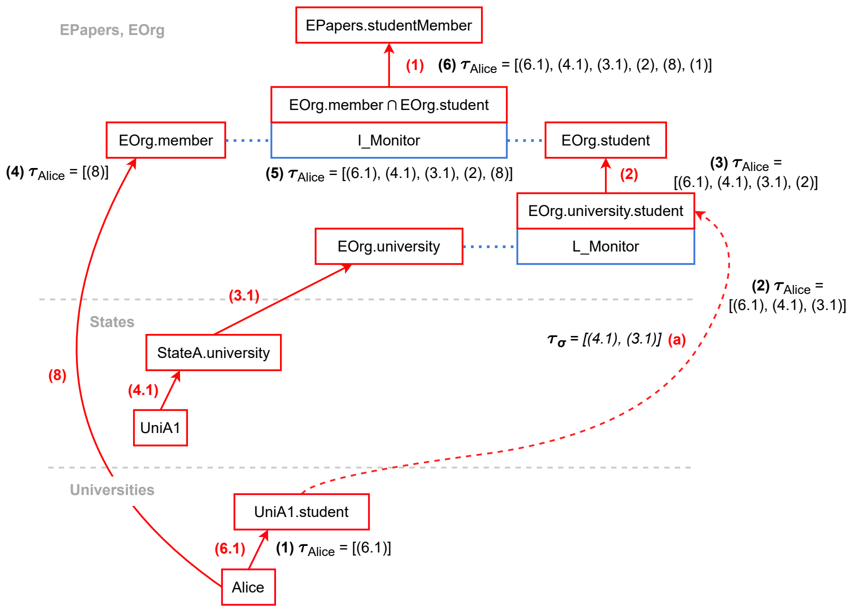


Fig. 7. Illustration of the construction of the proof concerning Alice. Red numbers on the edges represent the credentials of $\mathcal{P}_{EPapers}$ (Figure 4), while black numbers represent the evolution of the proof.

- Each time the algorithm propagates the solution by following a derived edge created by a L_monitor (see step 2 of the algorithm), the solution must include the support set $\sigma = (w_\sigma, \tau_\sigma)$ of such derived edge. The weight w_σ is the resulting weight updates of the credentials τ_σ. Hence, the solution (pi, w_{pi}, τ_{pi}) is updated to $(pi, w_{pi} * w_\sigma, \tau_{pi}.concat(\tau_\sigma))$. To avoid to rebuild the support set every time a derived edge is followed to propagate a solution, the algorithm stores in a data structure the support set of each edge when such edge is created.

For instance, executing *OFFchainBackwardSearch(EPapers.studentMember,* $\mathcal{P}_{EPapers}$*)* returns $S = \{(Alice, 1, \tau_{Alice}), \ldots\}$. We describe the construction of the proof τ_{Alice} beginning with the credential (6.1), which triggers a propagation of the solution representing Alice (the credential (8) also triggers the propagation and will be processed later in our example). In this specific case, the solution follows the derived edge (a) included to the proof graph as a result of the resolution of the linked inclusion, whose support set is composed by credentials (4.1) and (3.1) that are appended to the solution's proof. Afterwards, the solution follows the edge generated by the linked inclusion, credential (2), and the algorithm includes such edge to the proof. If credential (8) was already processed, the solution would activate the I_Monitor, otherwise the monitor will be activated when credential (8) will be processed. In either cases, when the I_Monitor is activated, the resulting proof will be the concatenation of the proof stored on the node $EOrg.student$ with the proof stored on the node $EOrg.member$. Finally, the credential (1) is appended to the proof when the solution follows the latest edge, and the propagation terminates on the node $EPapers.studentMember$ since the node has no outgoing edges. As a result, τ_{Alice} contains the edges computed from the credentials (6.1), (4.1), (3.1), (2), (8), (1), in this order. All weight computations are omitted because all of them are equal to 1 because the policy is an authorization policy (yes/no answer). Figure 7, derived from Figure 4, shows the construction of the proof τ_{Alice} in the proof graph.

4.3 Phase 3: verify a proof

Let (pi, w_{pi}, τ_{pi}) be a solution returned by the $OFFchainBackwardSearch(A.r, \mathcal{P})$ function computed by the off-chain module as previously described. We recall that the *verify* function is exposed by the on-chain module, which also stores the policy $\mathcal{P}$ used by the off-chain module to compute τ_{pi}. The execution of the *verify* function on τ_{pi}, *verify*(τ_{pi}), returns a tuple (p, r, w) indicating that, according to τ_{pi}, the principal p is member of the role r with weight equal to w.

If the input proof contains invalid credentials or credentials not belonging to the policy, i.e., it has been constructed in a wrong or malicious way, the *verify* function returns an empty solution. Otherwise, the result of the *verify* function is then used to perform role-based access control to regulate smart contracts' function execution as explained in Section 3, i.e., to check that p is actually the user who invoked the function execution, that r is the role required for executing such function, and that w is equal or greater than the minimum weight required. In the scenario of Figure 6, this check is executed by *EPapers ERC20 SC* doing: $EPapers.studentMember == r \land Alice == p \land w_{Alice} == 1$.

Let τ_{pi} be composed by the list of credentials $[c1, c2, \ldots, cn]$. The *verify* function iterates over τ_{pi}, it checks each credential ci actually belongs to the policy, and then it applies the function $\pi()$ (shown below) to ci. $\pi(ci)$ applies a rule depending on the type of ci, and it keeps an elaboration stack ρ, initially empty, to store intermediate results. Each element of the stack reflects an element stored in the proof graph that has been visited during the execution of the *OFFchainBackwardSearch* function, therefore it is represented as the triple (role, principal, weight). The only rule that increments the size of the stack is the one processing the simple member credential, which pushes onto the stack the new principals, while all the other rules pop at least an element from the stack and reduce, or keep unchanged, the size of the stack. Indeed, the first credential in a proof is always a simple member credential, otherwise $\pi()$ would fail.

The following system shows how each rule of $\pi(ci)$ works, showing on the right hand side the conditions, i.e., the type of the current credential and the value of the topmost element(s) extracted with the stack pop operation, and on the left hand side the effect, i.e., how the stack is changed. We use the term $\rho.pop().pop()$ as a shorthand to extract the two topmost elements of the stack, i.e., (a, b) = $\rho.pop().pop()$ where a = $\rho.pop()$ (1st), and b = $\rho.pop()$ (2nd).

$$
\pi(ci)\begin{cases}
\rho.push((A.r, pi, w)) & \text{if } ci == (A.r \longleftarrow^{w} pi)\\
\rho.push((A.r, pi, w * w1)) & \text{if } ci == (A.r \longleftarrow^{w} B.s) \land \rho.pop() == (B.s, pi, w1)\\
\rho.push((A.r, pi, w * w1 * w2)) & \text{if } ci == (A.r \longleftarrow^{w} B.s.t) \land\\
 & \rho.pop().pop() == ((B.s, C, w2), (C.t, pi, w1))\\
\rho.push((A.r, pi, w * min(w1, w2))) & \text{if } ci == (A.r \longleftarrow^{w} B.s \cap C.t) \land\\
 & \rho.pop().pop() ==\\
 & (((B.s, pi, w2), (C.t, pi, w1)) \lor ((C.t, pi, w1), (B.s, pi, w2)))\\
\perp & \text{otherwise}
\end{cases}
$$

The result returned by *verify* will be the topmost element of the stack. If $\pi(ci)$ returns $\perp$ the function *verify* immediately returns an empty element.

EPapers example: The execution steps of *verify*(τ_{Alice}) and the related stack states are illustrated in Figure 8. The proof verification requires 6 execution steps, and each of them is represented in Figure 8 by showing on the top the credential ci of the proof that is processed in that step, the stack state after applying $\pi(ci)$ (each box represents an element of the stack), and on the bottom the step number with the rule applied by $\pi(ci)$. In Step 6 where all the credentials in the

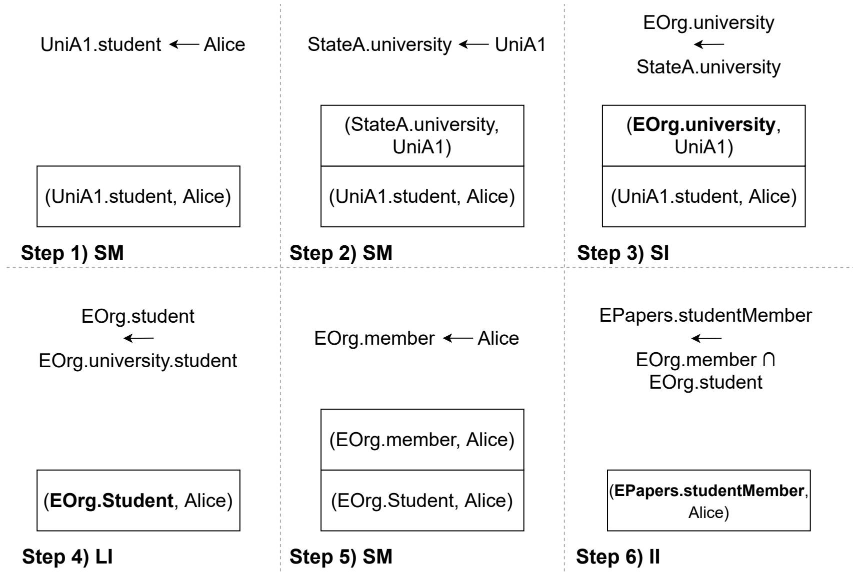


Fig. 8. Stack states during the execution of *verify*(τ_{Alice}). For each step the figure shows on the top the evaluated credential ci, the resulting stack state, and on the bottom the step number with rule applied by $\pi()$. The weights of credentials have been omitted since they are all equal to 1.

proof have been processed, the stack includes a single element, (Alice, EPapers.studentMember, 1), which is the result of the *verify* function. In this example, credential weights, whose value is 1 for all credentials, are omitted.

5 A PROTOTYPE OF A LAYER-2 DART

This section is aimed at evaluating the cost of executing the L2DART on-chain module in several distinct scenarios, to compare such costs with the ones of DART presented in [25]. The implementation is available at GitHub [27].

5.1 Prototype implementation

The on-chain module is implemented as a Solidity smart contract [19] that is executed on the Ethereum blockchain, while the off-chain module is implemented as a Python application which communicates with the on-chain module through the Web3py library [20]. Since this section is aimed at comparing the cost of executing DART and L2DART on the blockchain, in the following we focus on the L2DART Solidity implementation details that impact the most the gas cost. In Solidity, dynamic data structures must be stored in the persistent memory of a smart contract, called *storage*. Instead, static data structures can be memorized in the volatile memory, called *memory*. The cost of writing a data structure in *storage* is significantly higher than the cost of writing the same data structure in *memory* [52]. In DART, the backward search algorithm is executed on the blockchain. Since this algorithm requires to build the proof graph, whose size is unknown in advance, the implementation relies on mapping constructs that utilize the smart contract *storage*. In L2DART, instead, the backward search algorithm is executed by the off-chain module that, besides the solution, also returns an integer representing the amount of frames required to store all the triples during the execution of *verify* function onto the stack. This integer is passed to the *verify* function, along with the proof to be verified, and it allows the function to instantiate an array of fixed size that in Solidity can be stored in *memory*, thus reducing the amount of

	DART	L2DART
Role	43 659	43 786
Simple Member	72 194	87 556
Simple Inclusion	93 443	108 907
Linked Inclusion	136 672	131 441
Intersection Inclusion	138 357	154 130

Table 1. Gas cost storage of roles and credentials.

gas required. Intuitively, the upper-bound of that integer is the number of simple member credentials inside the proof τ_{pi}. Since the smart contract stores the role-expressions and the members inside Solidity *mappings*, credentials can be retrieved in O(1). Storage details are described in [25].

5.2 Experiments and comparison

The deployment of the L2DART smart contract has been observed to cost 2 073 852 units of gas, against 1 351 216 units of DART. This cost is proportional to the size of the bytecode that is uploaded on the blockchain. However, we recall that this is a one-time cost, which is paid only when the smart contracts are deployed on the blockchain. Table 1 shows the cost in gas to create a new role and to store trust credentials, which are comparable in the two implementations.

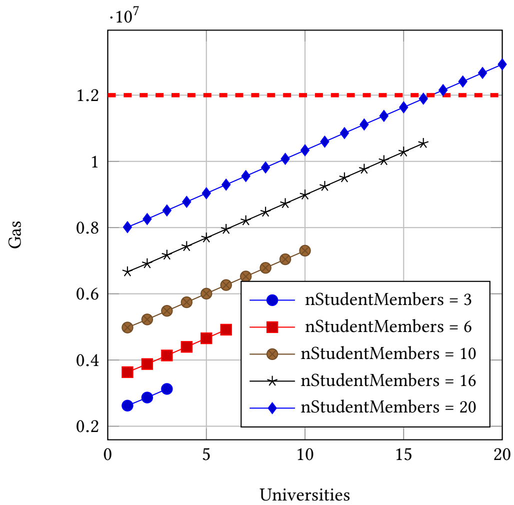
In the following of the section, we present three test scenarios comparing the cost of finding the solutions in DART against the cost of verifying them in L2DART. To simplify the description, we use the term *ONchainBackwardSearch* to identify the backward search algorithm executed on-chain by DART.

5.2.1 Test scenario A: Access control. **Description of the scenario:** We executed the *OFFchainBackwardSearch* to compute the members of the role $EPapers.studentMember$ of the policy $\mathcal{P}_{EPapers}$ applied in the access control scenario described in Section 3.2, and we evaluated the cost to execute the *verify* function of the L2DART smart contract on the corresponding proofs (operation 3.2 in Figure 6). We then compared such costs with the cost of finding the members of the same role according to the same policy obtained by executing *ONchainBackwardSearch* presented in [25].

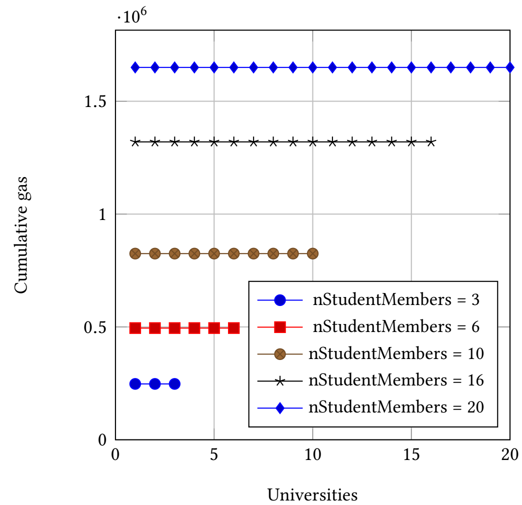
Description of the experiment and results: Figure 9(a) shows the gas consumed to execute *ONchainBackwardSearch* to find the members of the role $EPapers.studentMember$ [25]. The experiments were conducted varying the number of *Universities* and the number of principals (*nStudentMembers*) holding to the role $EPapers.studentMember$ similarly to Alice in Figure 4. The dashed horizontal red line indicates the block gas limit of Ethereum at the time we conducted the experiments [25], i.e., 12M gas units.

Figure 9(b), instead, shows the sum of the costs to **verify all the proofs** related to the solutions found by the *OFFchainBackwardSearch*. For instance, each point of the plot for *nStudentMembers*=3 shows the cost to verify 3 proofs. Obviously, the solution sets returned by the two backward search implementations are the same. Comparing the results in Figures 9(a) and 9(b) we notice that the cost to execute *verify* is much lower than the cost of executing the *ONchainBackwardSearch*. Taking as example the test case with 20 universities and *nStudentMembers* equal to 20, we observe that the execution of *ONchainBackwardSearch* costs about 12 931 738 units of gas, which exceeds the block gas limit, while the execution of the *verify* function on the corresponding 20 proofs costs about 1 649 872 units of gas, i.e., almost 8 times less, as shown on the topmost plot of Figure 9(b).

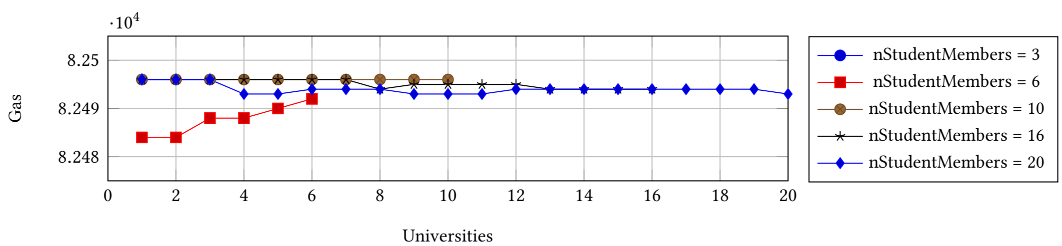
Since in the access control scenario described in Section 3.2 the smart contract EPapers ERC20 SC needs to verify one proof only (i.e., Alice's one, see step 3.2 of Figure 6), Figure 9(c) shows the average gas consumed to execute the *verify*



(a) DART, cost in gas to execute *ONchainBackwardSearch* [25].



(b) L2DART, sum of the costs in gas to verify all the $nStudentMembers$ proofs produced by the $OFFchainBackwardSearch$.



(c) L2DART, average cost in gas of a single *verify* execution.

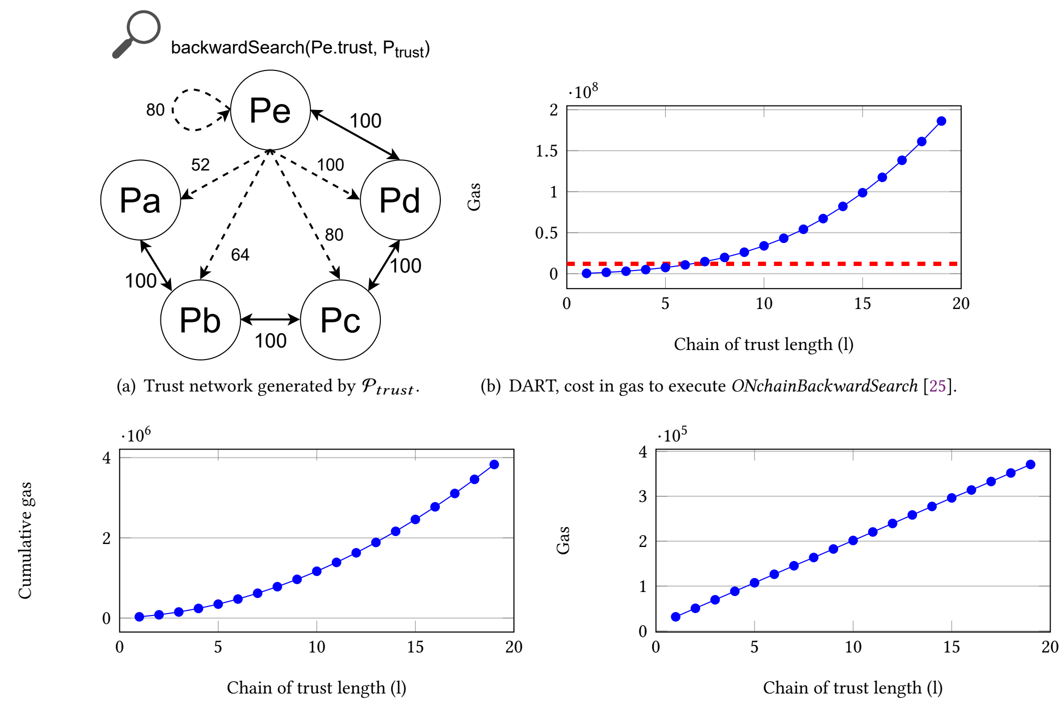
Fig. 9. Comparison of cost in gas of test scenario A, access control, varying number of student members and universities.

function **on a single proof**. For example, the average cost of executing the $verify$ function on a single proof in the case of 6 *nStudentMembers* and 6 *Universities* is 82 492 units of gas, with a variance of 2.67 units of gas. The cost is almost constant because for each principal the set of credentials in the proof to demonstrate the role *EPapers*.$studentMember$ does not depend on the number of universities or student members present in the policy.

Discussion of the results: The tests show the cost to verify the proofs of the solutions computed off-chain is much lower than finding them directly on-chain, especially if the use case requires to verify a single solution. Note that in our tests the cost to verify a proof is almost constant and equal for all the student members because the set of credentials held by each student member is similar to credential set of Alice, as shown in Section 2.2.4. The goal of the experiment is to show that the verification cost does not necessarily grow with the number of credentials in the policy, because it depends on the length of the proof. This allows to adopt L2DART in real access control scenarios. Finally, we delve into the underlying causes of the variations in gas consumed by the verify function on a proof related to the *OFFchainBackwardSearch* (i.e., see Figures 9(c) and 11(c)). In particular, we investigate the source of these small variations by using Sol-Profiler[2], a library providing line-by-line gas usage of solidity smart contracts. The profiler results generated by the tool reveal that such variations in gas are due to the functions' arguments allocated in the Calldata area. In this memory area, the gas cost for allocating non-zero bytes is higher than the gas cost for allocating zero bytes.

5.2.2 Test scenario B: Web of trust. **Description of the scenario:** This scenario takes into account a policy describing a generic trust network [25], where each principal *pi* defines their *trust* relations with the others directly, using simple member trust credentials (credential T1), and indirectly, using linked inclusion trust credentials (credential T2):

$$(T1)\; pi.trust \xleftarrow{w_1} pj \qquad (T2)\; pi.trust \xleftarrow{w_2} pi.trust.trust$$



(a) Trust network generated by $\mathcal{P}_{trust}$.

(b) DART, cost in gas to execute *ONchainBackwardSearch* [25].

(c) L2DART, cumulative sum of the costs in gas. For each point X=l the Y axis is equal to $\sum_1^l cost(verify(\tau))$.

(d) L2DART, cost in gas of a single *verify* execution.

Fig. 10. Comparison of cost in gas of test scenario B, web of trust.

[2] https://www.npmjs.com/package/@0x/sol-profiler

Figure 10(a) shows a trust network involving 5 principals generated by the following policy $\mathcal{P}_{trust}$:

$$(T2.1) \quad Pb.trust \leftarrow^{80} Pb.trust.trust; \quad (T1.1) \quad Pb.trust \leftarrow^{100} Pa; \quad (T1.5) \quad Pa.trust \leftarrow^{100} Pb$$

$$(T2.2) \quad Pc.trust \leftarrow^{80} Pc.trust.trust; \quad (T1.2) \quad Pc.trust \leftarrow^{100} Pb; \quad (T1.6) \quad Pb.trust \leftarrow^{100} Pc$$

$$(T2.3) \quad Pd.trust \leftarrow^{80} Pd.trust.trust; \quad (T1.3) \quad Pd.trust \leftarrow^{100} Pc; \quad (T1.7) \quad Pc.trust \leftarrow^{100} Pd$$

$$(T2.4) \quad Pe.trust \leftarrow^{80} Pe.trust.trust; \quad (T1.4) \quad Pe.trust \leftarrow^{100} Pd; \quad (T1.8) \quad Pd.trust \leftarrow^{100} Pe$$

Executing the backward search algorithm on $\mathcal{P}_{trust}$ to find the members of the role $Pe.trust$, the solution set will be composed by the following elements: $\{(Pd, 100), (Pe, 80), (Pc, 80), (Pb, 64), (Pa, 52)\}$ with τ_{Pa} being (T1.4), (T1.3), (T1.2), (T1.1), (T2.4), (T2.4), (T2.4). Figure 10(a) shows the network with the dashed arrows representing the solutions, while the plain arrows representing the credentials of type T1 (the related weights are shown next to the arrows).

Description of the experiment and results: We created a set of trust networks whose topologies are similar to the one shown in Figure 10(a), and whose length of the longest chain of trust reachable from Pe ranges from 1 to 19.

Figure 10(b) shows the cost in gas to execute *ONchainBackwardSearch(Pe.trust,* $\mathcal{P}^{l}_{trust}$*)* (corresponding to the "active trust network" in [25]) varying the length l of the longest chain of trust reachable from Pe in the trust network produced by the policy $\mathcal{P}^{l}_{trust}$. Figure 10(b) shows that the gas consumed is very high, overcoming the current Ethereum block gas limit (12M units, represented by the horizontal dashed red line) when $l = 7$. Figure 10(c), instead, shows the sum of the costs in gas to **verify all the proofs** related to the solutions produced by the execution of *OFFchainBackwardChain*. Similarly to the scenario A, the cost in gas to verify all the proofs using the *verify* function of L2DART is much lower than the cost to compute the solutions using the *ONchainBackwardSearch* function of DART. For instance, choosing $l = 19$, the on-chain cost of computing the solutions is about 53 times more than the cost of verifying all the related proofs. Finally, Figure 10(d) shows the gas consumed to **verify a single proof**: verifying a proof of length 1 requires about 31 000 units of gas, and adding one credential to the proof increases the verification cost of about 19 000 units of gas. Hence, supposing a block gas limit of 12M units, it is possible to verify proofs having lengths up to 60 credentials, i.e., involving a principal distant 60 steps in the trust network.

Discussion of the results: The experiments conducted in the Web of Trust scenario show us that combining on-chain storage and off-chain computation is feasible to process a trust network while preserving the computational auditability of a blockchain, and the applicability of L2DART to scenarios not focused only on access control.

5.2.3 Test scenario C: Book Recommendation System. **Description of the scenario:** In this scenario (derived from [50]), a student (*Alice*) trusts the recommendations from the reviewers of the *RecSys* recommendation system, which specifies that only the participants having the role of *buyer* and *expert* are able to review an item, and the role of *expert* is given to professors of several recognized universities. This is represented by the following policy $\mathcal{P}_{Rec}$:

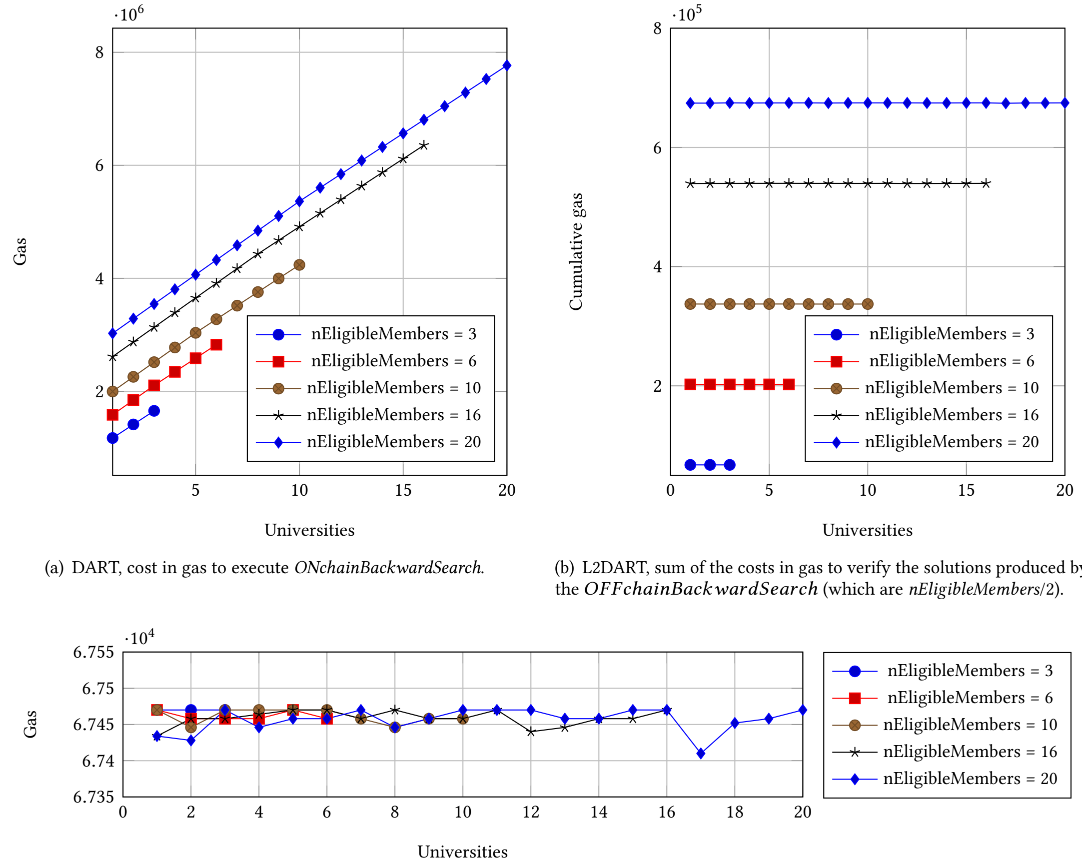
$$Alice.recommendationFrom \leftarrow^{1.0} RecSys.reviewer$$

$$RecSys.reviewer \leftarrow^{1.0} RecSys.expert \cap RecSys.buyer$$

$$RecSys.expert \leftarrow^{1.0} RecSys.university.professor$$

$$RecSys.university \leftarrow^{1.0} StateA.university$$

To complete policy $\mathcal{P}_{Rec}$, we need to add the simple member credentials that assign the role of *professor* and *buyer* to principals. We denote with $\mathcal{P}^{n,m}_{Rec}$ a policy derived from policy $\mathcal{P}_{Rec}$ which selects n Eligible Members (i.e., principals) as professor of m different universities and randomly assigns also the role of *buyer* to half of them.



(a) DART, cost in gas to execute *ONchainBackwardSearch*.

(b) L2DART, sum of the costs in gas to verify the solutions produced by the $OFFchainBackwardSearch$ (which are *nEligibleMembers*/2).

(c) L2DART, average cost in gas of a single *verify* execution.

Fig. 11. Comparison of cost in gas of test scenario C, Recommendation System, varying number of eligible members and universities.

Description of the experiment and results: We set up the experiment to vary the number of *Universities*, m, in the range [1,..,20], and the number of principals involved in the policy, $nEligibleMembers$, in the set $\{3, 6, 10, 16, 20\}$. Figure 11(a) shows that the amount of gas to execute the *ONchainBackwardSearch* function to compute the set of reviewers trusted by *Alice* according to $\mathcal{P}_{Rec}^{n,m}$ depends on both the number of universities and the number of eligible members in the policy. The computation of the results does not exceed the block gas limit (of 12M units) and it achieves the highest cost (7 767 083 units) when both the number of universities and the number of eligible reviewers are equal to 20. Figure 11(b), instead, shows the total amount of gas necessary to execute the *verify* function on all the proofs related to the solutions produced by *OFFchainBackwardSearch*, i.e., the set of recommended reviewers trusted by *Alice*, while Figure 11(c) shows the average cost to verify only one of these proofs. Similarly to the scenario A, the amount of gas consumed by L2DART to verify one solution is constant with respect to the number of universities and *nEligibleMembers*. Indeed, the cost mainly depends on the length of the path connecting the solution to the specific role in the proof graph. Summarizing, from Figures 11(a) and 11(b) we observe that the gas cost of computing the solutions with *ONchainBackwardSearch* is about one order of magnitude greater than the total amount of gas required to execute the *verify* functions on the same solutions.

Discussion of the results: As for the other two scenarios, the cost in terms of gas of executing the *verify* function is much lower than the cost of executing the *ONchainBackwardSearch* function, therefore we derive the same conclusions.

6 DISCUSSION

This section discusses relevant aspects of the proposed approach and presents a security analysis of L2DART.

Evaluation. We evaluate whether L2DART is a proper Layer-2 architecture implementing the verifiable computation protocol as derived from Eberhardt et al. [16] and described in Section 2.1, and we assess the design properties presented in Section 3.2. L2DART implements the off-chain computation mode: it outsources the heavy execution task, the computation of the members of a role, to a node external the blockchain while the data, i.e., the policy, is entirely stored on-chain. The L2DART approach also complies with the verifiable computation protocol [16]: *i)* L2DART is non-interactive, i.e., it requires only one message, the proof, from the Prover (Alice) to the Verifier (e.g. the ERC20 SC in Figure 6); *ii)* the verification of a solution is much cheaper compared to compute it with the backward search algorithm; *iii)* L2DART does not have any strong security assumptions on the Prover, since it is assumed to be untrusted, while the Verifier is trusted being implemented on the blockchain; *iv)* since all the data is on the blockchain, no private inputs are required, this particular implementation of L2DART does not need zero-knowledge properties. Moreover, we derive that L2DART ensures P1, Data auditability, because the policy is stored on-chain; it ensures P2, Computational auditability, because it is possible to dispute the off-chain module about the correctness of the results; it ensures P3, Affordable Fees, because storing the roles and credentials, and the on-chain proof verifications are cheap in terms of gas in our experiments. In particular, the complexity of the code of the backward search algorithm is cubic with respect to the number of credentials in a policy [36], while the complexity of the proof verification algorithm is linear with respect to the number of credentials in the proof. The proof length depends on the policy and, due to the activation of the monitors *I_Monitor* and *L_Monitor* (see Section 2.2.3), a subset of credentials of the policy could be replicated in the proof. As a result, the number of credentials in the proof might be larger than the number of credentials in the policy. On the other hand, the verification algorithm iterates over the credentials of the proof, and each step is computationally light, because it simply applies the rule $\pi()$ to one credential of the proof, as described in Section 4.3. Moreover, we notice that, in general, a proof does not necessarily contain all the credentials in the policy. For instance, if we take into account the example of the Test scenario A shown in Section 5.2.1, the length of the proof required to prove that Alice holds the role $Epapers.studentMember$ is constant, i.e., it consists of 6 credentials (see Figure 7), regardless of the number of students and of universities, i.e., of the credentials representing them, in the policy (see Section 2.2.4). The same applies for the test scenario C (see Section 5.2.3). In the example described by the Test scenario B (see Section 5.2.2), whose policy $\mathcal{P}_{trust}$ represents a trust network, the length of a proof depends on the length l of the trust chain that connects the two principals. Indeed, a proof includes l credentials of type T1, and $l-1$ copies of a credential of type T2. Hence, the length of a proof is about twice the length of the trust chain, and the length of the latter is always less than the length of the policy by construction of $\mathcal{P}_{trust}$. For instance, in the trust network shown in Figure 10(a), the trust chain between Pe and Pa involves 4 principals including Pa, i.e., $l = 4$. In this case, the length of the proof τ_{Pa} (shown in Section 5.2.2) is 7 because it refers to a trust chain with $l = 4$ and the credential T2.4 is replicated 3 times, while the policy has 12 credentials. Finally, we also observe that, since the current most popular blockchains impose constraints on the amount of computation that a smart contract can use in a transaction, the comparison among the backward search and the proof verification algorithms must take into account such constraints that limit the maximum number of credentials that such algorithms can process. In this respect, the experiments we conducted in Section 5.2 have shown that, within these limits, in the selected scenarios the cost of computing the solutions is always considerably larger than the cost of verifying even all the related proofs.

Security analysis. In order to analyze the security of the proposed system, we consider the scenario of Figure 6 where *Epapers* deploys a smart contract *EPapers ERC20 SC* whose functions can be executed only by users having the role $Epapers.studentMember$ according to the policy $\mathcal{P}_{EPapers}$ stored by the on-chain module. Hence, the smart contract *EPapers ERC20 SC* requires as input parameter a proof τ proving that the user invoking it holds the role $Epapers.studentMember$, and it calls the *verify* function of the L2DART on-chain module in order to validate such proof. In particular, the on-chain module computes the role granted by the proof τ, and *EPapers ERC20 SC* checks that such role is equal to $Epapers.studentMember$ and that it is granted to the user who is invoking *EPapers ERC20 SC*.

In our scenario, *Alice* is a user who would like to exploit the functions of the smart contract *EPapers ERC20 SC*. *Alice* must provide to *EPapers ERC20 SC* a proof τ (normally generated by the off-chain module) which grants her the role $Epapers.studentMember$ according to the policy $\mathcal{P}_{EPapers}$. Both *Epapers* and *Alice*, as well as the other participants of the system, are able to interact with the on-chain module which is trusted since it is deployed on a public blockchain. Instead, the off-chain module is an untrusted component, since it is executed on the local device of *Alice*. As a matter of fact, *Alice* could alter the off-chain component (or even user another tool) to generate malicious proofs. Therefore, the participants of the system can behave maliciously by generating the following attacks.

- **Attack 1: *Alice* tries to execute the smart contract *EPapers ERC20 SC* submitting a valid proof that, however, does not grant her the role *Epapers.studentMember*.** The attack is conducted as follows. *Alice* sends to *EPapers ERC20 SC*'s smart contract a correct proof τ proving, however, that *Alice* holds the role $Epapers.staffMember$ (instead of $Epapers.studentMember$). *EPapers ERC20 SC*'s smart contract calls the *verify* function of the on-chain module to verify τ with respect to the current policy $\mathcal{P}_{EPapers}$. The verify function navigates the credentials of the policy and returns that τ proves that *Alice* holds the role $Epapers.staffMember$. The soundness property (see Section 2.2.3) of the proof graph ensures that the proof P resolves to a path representing a valid solution. Since the role found as result of the execution of the *verify* function is not the one required for the execution of *EPapers ERC20 SC*'s functions, the smart contract *EPapers ERC20 SC* denies the execution request received from *Alice*. Similarly, if *Alice* submits to *EPapers ERC20 SC*'s smart contract a valid proof τ proving, however, that another user (say *Bob*) holds the role $Epapers.studentMember$ (instead of *Alice*), the smart contract *EPapers ERC20 SC* would deny the execution request received from *Alice* as well, because the proof does not assign any role to her.
- **Attack 2: *Alice* tries to execute the smart contract *EPapers ERC20 SC* submitting a not valid proof that pretends to grant her the role *Epapers.studentMember*.** The attack is conducted as follows. *Alice* sends to *EPapers ERC20 SC*'s smart contract an incorrect proof τ' which pretends that *Alice* holds the role $Epapers.studentMember$. For instance, τ' could include a credential that is not part of the policy $\mathcal{P}_{EPapers}$, or it can be incorrectly formatted, or does not correspond to a valid statement. The smart contract *EPapers ERC20 SC* invokes the on-chain module to execute the *verify* function and, as a result, it is notified that τ' is not valid according to the current policy. Consequently, the smart contract *EPapers ERC20 SC* does not allow the execution of the function requested by *Alice*.

- **Attack 3: *Epapers* tries to deny the access to the smart contract *EPapers ERC20 SC* to *Alice*, although *Alice* submits a correct proof that she holds the role *Epapers.studentMember*.** The attack is conducted as follows. *Alice* submits to the *EPapers ERC20 SC*'s smart contract a valid proof τ that she holds the role $Epapers.studentMember$ according to the current policy, which can be verified by using the verify function of the on-chain module. The completeness property (see Section 2.2.3) of the proof graph ensures that if τ holds the role $Epapers.studentMember$, then there exists a path in the proof graph that can be used to prove it. Even if the proof is correct and states that *Alice* holds the role $Epapers.studentMember$, *Epapers* unduly denies the execution of the smart contract by *Alice*, pretending that the verification process has failed. Because of the auditability property provided by the framework, *Alice* (and also other participants) can reliable detect the misbehavior of *Epapers*. Indeed, the code of *Epapers*'s smart contract, the code of the L2DART on-chain module, as well as the current policy are available on the blockchain. In addition, since every correct communication between *Alice* and the smart contract *EPapers ERC20 SC* results in a verifiable and permanent transaction stored on the blockchain, this would serve as a proof to demonstrate the status of the policy at specific point in time.

Finally, any man-in-the-middle attempt on the channel used by the off-chain module to interact with blockchain nodes can be mitigated by using secured channels that provide confidentiality, integrity, and authenticity. The former property can be provided by Transport Layer Security (TLS) protocols, while the latter two properties are provided by digital signatures. Indeed, any blockchain transaction, which writes data, is always digitally signed.

Drawbacks. Besides the important advantages discussed in the previous sections, layer-2 technologies also introduce some drawbacks. First of all, blockchain layer-2 technologies rely on resources that do not benefit from the same security and decentralization as the layer 1 nodes participating in the blockchain consensus. As a matter of fact, off-chain nodes might be more susceptible to attacks, being composed by a much smaller network of nodes or even a single node, and might require higher trust assumptions depending on who has the governance of such nodes. Moreover, designing a two layered blockchain application typically requires more effort than designing a blockchain application. Furthermore, sometime also the modification of the first layer is required. For example, to deploy the Lightning Network Bitcoin had to perform the SegWit soft fork [7]. Finally, the fact that some blocks could be produced after the off-chain module has read the data needed for the off-chain computation on the blockchain, but before that the result is verified on the blockchain, might introduce a race condition problem. For instance, such race condition issue could affect L2DART. In particular, a L2DART policy $\mathcal{P}$ could be modified between the time T1 when a proof τ_u for a user u through the *OFFchainBackwardSearch* function has been produced and the time T2 > T1 when the proof will be verified to access to a smart contract c. As a consequence, the verification of the proof τ_u, which was valid at time T1 might fail at time T2 and the execution right could be denied. Although troublesome, denying an access due to these race conditions is actually the correct behavior. Hence, the users would have to produce a new proof with *OFFchainBackwardSearch* with the updated policy, and to ask to execute the smart contract c again by submitting this new proof. However, we note that this problem arises also in a full on-chain implementation due to a vulnerability called *transaction-ordering dependence* [37], i.e., when the outcome of two transactions Tx1 and Tx2 depends on their ordering inside a block.

7 RELATED WORK

Blockchain-based access control mechanisms have been applied to IOT [3, 40], healthcare [46], and cloud storage [23]. However, the implementations typically involve direct role and attribute assignments, which can be difficult to apply to trans-organizational scenarios where different entities assign roles that, once combined, can infer other roles following a specific set of rules. This section describes a number of existing blockchain-based access control system implementations not specialized to any use case and in Table 2 we compare the characteristics of such solutions against the approach we propose in this paper.

OpenZeppelin [39] is a popular framework that provides a solid implementation of the RBAC model in Solidity for Ethereum where users' roles are stored on-chain. In particular, each role is stored on-chain in a map data structure holding the list of accounts with that role. OpenZeppelin does not support role inference, because users are only able to assign a specific role to an Ethereum account and smart contract functions check if the account has been assigned a specific role.

Cruz et al. [13] propose a challenge-response based implementation of a RBAC in a trans-organizational environment. Roles are stored on-chain by using a smart contract and each role is assigned by an institution to a blockchain address belonging to a user. A service provider asks the user to prove they control an address that holds a specific role. A challenge-response protocol is executed between the user and the service provider: the service provider sends a message to the user, and the user must sign the message with the private key that generated the address associated to the role the user is claiming to have; if the user sends back a valid signature, the service provider is sure the user holds the role they claim to have.

Di Francesco Maesa et al. [15] codify attribute-based access control XACML policies in smart contracts in order to benefit from blockchain auditability and easily identify misbehavior from one of the parties. In this approach, users' roles are represented as attributes. However, this approach does not allow users to make inference over attributes and off-chain computation is not executed because policies are evaluated on-chain by the related smart contracts.

Summarizing, the main similarity among all the approaches listed in Table 2 is that they allow their users to define their roles and to assign them to other users (in DART and L2DART, using Simple member credentials, see Section 2.2.1), they store the credentials representing the roles on the blockchain, and they use them in the access control process to decide whether to grant a privilege to a user or not. However, DART and L2DART differ from the other approaches because they allow their users to define their trust relations through Simple inclusion, Linked inclusion, and Intersection inclusion credentials (see Section 2.2.1). Consequently, the access control process of DART and L2DART uses both the roles that have been directly assigned to the users, as well as the roles that have been inferred exploiting the trust relations (executing the chain discovery algorithm, see Section 2.2.3, to perform role inference). Hence, the main advantage of DART and L2DART with respect to the other approaches is the flexibility of not requiring to explicitly assign all the roles to all the users. As a matter of fact, exploiting the RT framework, DART and L2DART allow their users to define their roles indirectly, taking into account the roles defined by other users they trust. This is useful, for instance, in trans-organizational scenarios where roles are assigned by multiple cooperating organizations. Finally, the main advantage of L2DART with respect to DART is that the latter executes the role inference algorithm on-chain, while the former executes it off-chain, and produces proofs of the inferred roles that are validated. Hence, L2DART mitigates the scalability problem of DART, as clearly shown by the results of the experiments we conducted, described in Section 5.2, while maintaining blockchain data and computational auditability.

8 CONCLUSION AND FUTURE WORK

In this paper we proposed L2DART, a Role Based Trust Management System implemented on top of a public and permissionless blockchain allowing to infer the roles held by each of its users from the direct and indirect trust relationships they expressed. L2DART overcomes the limitations of its predecessor, DART, by adopting the off-chain computation model. In particular, L2DART takes advantage of a verifiable computation protocol to split the role

Proposals	Credential storage	Role inference	Off-chain computation & on-chain verification
OpenZeppelin [39]	on-chain	✗	✗
Cruz et al. [13]	on-chain	✗	✗
Di Francesco Maesa et al. [15]	on-chain	✗	✗
DART [25]	on-chain	✓	✗
L2DART	on-chain	✓	✓

Table 2. Comparison of existing proposals.

calculation process in two different steps: one that is executed off-chain, i.e., the proof computation, and the other that is executed on-chain, the proof verification.

A prototype we implemented on Ethereum shows that, gas wise, the verification of the proof has a cost much lower than computing the solutions directly on chain. At the same time, L2DART ensures data and computational auditability required by a blockchain-based trust management system. Hence, the cost of executing L2DART on the blockchain is affordable, thus allowing L2DART to be successfully deployed in the real world implementing use cases such as supporting customers to identify trustworthy service providers [45, 48], or to certify some properties of users (such as their identities, position, etc…) [9, 30]. Although our prototype has been implemented on Ethereum, the on-chain computation of L2DART can be implemented by any blockchain whose smart contracts support array and key-value data structures to store and retrieve roles, credentials, and proofs required to apply the rules to verify a proof. Examples are EOS.IO, Hyperledger Fabric, and any system based on the Ethereum Virtual Machine, such as Quorum, Hyperledger Besu, and Polygon.

As future improvements, L2DART can be naturally extended to support more RT credentials and other chain discovery algorithms. Moreover, the current version of the on-chain verification algorithm does not store intermediate solutions, meaning that the same role could be re-computed multiple times. Therefore, the on-chain verification algorithm could be adapted to re-use intermediate solutions in those scenarios that would reduce the on-chain costs. Moreover, the L2DART design could be adapted to integrate privacy preserving techniques to support access control scenarios involving sensible user data. Finally, starting from the experience acquired with this research about layer-2 technologies and off-chain computation, we plan to define a framework to help system designers to design their layer-2 decentralized applications. The analysis of the set of requirements related to the application (such as scalability, cost, and privacy needs) provided by the framework will help the designer in choosing the right layer-2 technologies that best satisfy the requirements, e.g., to understand when an on-chain operation could be outsourced off-chain for reducing the execution cost.

REFERENCES

[1] Alfarez Abdul-Rahman and Stephen Hailes. 1997. A distributed trust model. In *Proceedings of the 1997 Workshop on New Security Paradigms, Langdale, Cumbria, United Kingdom, September 23-26, 1997*, Tom Haigh, Bob Blakley, Mary Ellen Zurko, and Catherine Meodaws (Eds.). ACM, 48–60. https://doi.org/10.1145/283699.283739

[2] Rishav Raj Agarwal, Dhruv Kumar, Lukasz Golab, and Srinivasan Keshav. 2020. Consentio: Managing Consent to Data Access using Permissioned Blockchains. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2020, Toronto, ON, Canada, May 2-6, 2020*. IEEE, 1–9. https://doi.org/10.1109/ICBC48266.2020.9169432

[3] Tejasvi Alladi, Vinay Chamola, Reza M. Parizi, and Kim-Kwang Raymond Choo. 2019. Blockchain Applications for Industry 4.0 and Industrial IoT: A Review. *IEEE Access* 7 (2019), 176935–176951. https://doi.org/10.1109/ACCESS.2019.2956748

[4] Claudio A. Ardagna, Ernesto Damiani, Sabrina De Capitani di Vimercati, Sara Foresti, and Pierangela Samarati. 2007. Trust Management. In *Security, Privacy, and Trust in Modern Data Management*, Milan Petkovic and Willem Jonker (Eds.). Springer, 103–117. https://doi.org/10.1007/978-3-540-

69861-6_8

[5] Arpachain. 2018. *ARPA Whitepaper.* https://docsend.com/view/t69gzij [Online, accessed 21 October 2021].

[6] Stefano Bistarelli, Fabio Martinelli, and Francesco Santini. 2008. A semantic foundation for trust management languages with weights: An application to the RT family. In *International Conference on Autonomic and Trusted Computing*. Springer, 481–495.

[7] Bitcoin Wiki. 2021. *Segregated Witness.* https://en.bitcoin.it/wiki/Segregated_Witness [Online, accessed 21 October 2021].

[8] Matt Blaze, Joan Feigenbaum, and Jack Lacy. 1996. Decentralized Trust Management. In *1996 IEEE Symposium on Security and Privacy, May 6-8, 1996, Oakland, CA, USA*. IEEE Computer Society, 164–173. https://doi.org/10.1109/SECPRI.1996.502679

[9] Ying Cai and Qinglong Wang. 2010. Certificates Distributed Storage Scheme in the Trust Management System. In *Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2010), Darmstadt, Germany, 15-17 October, 2010, Proceedings*, Isao Echizen, Jeng-Shyang Pan, Dieter W. Fellner, Alexander Nouak, Arjan Kuijper, and Lakhmi C. Jain (Eds.). IEEE Computer Society, 490–493. https://doi.org/10.1109/IIHMSP.2010.125

[10] Anamika Chauhan, Om Prakash Malviya, Madhav Verma, and Tejinder Singh Mor. 2018. Blockchain and Scalability. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS Companion 2018, Lisbon, Portugal, July 16-20, 2018*. IEEE, 122–128. https://doi.org/10.1109/QRS-C.2018.00034

[11] Weili Chen, Tuo Zhang, Zhiguang Chen, Zibin Zheng, and Yutong Lu. 2020. Traveling the token world: A graph analysis of Ethereum ERC20 token ecosystem. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 1411–1421. https://doi.org/10.1145/3366423.3380215

[12] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. 2015. *Secure multiparty computation.* Cambridge University Press.

[13] Jason Paul Cruz, Yuichi Kaji, and Naoto Yanai. 2018. RBAC-SC: Role-Based Access Control Using Smart Contract. *IEEE Access* 6 (2018), 12240–12251. https://doi.org/10.1109/ACCESS.2018.2812844

[14] Marcin Czenko, Sandro Etalle, Dongyi Li, and William H. Winsborough. 2007. An Introduction to the Role Based Trust Management Framework RT. In *Foundations of Security Analysis and Design IV, FOSAD 2006/2007 Tutorial Lectures*, Alessandro Aldini and Roberto Gorrieri (Eds.). Lecture Notes in Computer Science, Vol. 4677. Springer, 246–281. https://doi.org/10.1007/978-3-540-74810-6_9

[15] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. 2019. A blockchain based approach for the definition of auditable Access Control systems. *Comput. Secur.* 84 (2019), 93–119.

[16] Jacob Eberhardt and Jonathan Heiss. 2018. Off-chaining Models and Approaches to Off-chain Computations. In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, SERIAL@Middleware 2018, Rennes, France, December 10, 2018*. ACM, 7–12. https://doi.org/10.1145/3284764.3284766

[17] Jacob Eberhardt and Stefan Tai. 2017. On or Off the Blockchain? Insights on Off-Chaining Computation and Data. In *Service-Oriented and Cloud Computing - 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10465)*, Flavio De Paoli, Stefan Schulte, and Einar Broch Johnsen (Eds.). Springer, 3–15. https://doi.org/10.1007/978-3-319-67262-5_1

[18] Jacob Eberhardt and Stefan Tai. 2018. ZoKrates - Scalable Privacy-Preserving Off-Chain Computations. In *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), iThings/GreenCom/CPSCom/SmartData 2018, Halifax, NS, Canada, July 30 - August 3, 2018*. IEEE, 1084–1091. https://doi.org/10.1109/Cybermatics_2018.2018.00199

[19] Ethereum. 2021. *Solidity Documentation.* https://docs.soliditylang.org/en/v0.8.9/index.html [Online, accessed 21 October 2021].

[20] Ethereum. 2021. *Web3py Documentation.* https://web3py.readthedocs.io/en/stable/ [Online, accessed 21 October 2021].

[21] Ethereum Optimism. 2020. *On To New Beginnings.* https://medium.com/plasma-group/on-to-new-beginnings-e9d76b170752 [Online, accessed 21 October 2021].

[22] Davide Fais, Maurizio Colombo, and Aliaksandr Lazouski. 2009. An Implementation of Role-Base Trust Management Extended with Weights on Mobile Devices. *Electron. Notes Theor. Comput. Sci.* 244 (2009), 53–65. https://doi.org/10.1016/j.entcs.2009.07.038

[23] Md. Sadek Ferdous, Andrea Margheri, Federica Paci, Mu Yang, and Vladimiro Sassone. 2017. Decentralised Runtime Monitoring for Access Control Systems in Cloud Federations. In *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017*, Kisung Lee and Ling Liu (Eds.). IEEE Computer Society, 2632–2633. https://doi.org/10.1109/ICDCS.2017.178

[24] David Ferraiolo, D Richard Kuhn, and Ramaswamy Chandramouli. 2003. *Role-based access control.* Artech house.

[25] Luca Franceschi, Andrea Lisi, Andrea De Salve, Paolo Mori, and Laura Ricci. 2021. DART: Towards a role-based trust management system on blockchain. In *30th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2021, Bayonne, France, October 27-29, 2021*. IEEE, 75–80. https://doi.org/10.1109/WETICE53228.2021.00025

[26] GitHub. 2020. DART. https://github.com/0Alic/DART [Online, accessed 21 October 2021].

[27] GitHub. 2021. *L2DART.* https://github.com/0Alic/DART-estensione [Online, accessed 21 October 2021].

[28] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2020. SoK: Layer-Two Blockchain Protocols. In *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12059)*, Joseph Bonneau and Nadia Heninger (Eds.). Springer, 201–226. https://doi.org/10.1007/978-3-030-51280-4_12

[29] Maurice Herlihy. 2018. Atomic Cross-Chain Swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, Calvin Newport and Idit Keidar (Eds.). ACM, 245–254. https://dl.acm.org/citation.cfm?id=3212736

[30] Amir Herzberg, Yosi Mass, Joris Mihaeli, Dalit Naor, and Yiftach Ravid. 2000. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*. IEEE Computer Society, 2–14. https://doi.org/10.1109/SECPRI.2000.848442

[31] Maxim Jourenko, Kanta Kurazumi, Mario Larangeira, and Keisuke Tanaka. 2019. SoK: A Taxonomy for Layer-2 Scalability Related Protocols for Cryptocurrencies. *IACR Cryptol. ePrint Arch.* (2019), 352. https://eprint.iacr.org/2019/352

[32] Soohyeong Kim, Yongseok Kwon, and Sunghyun Cho. 2018. A Survey of Scalability Solutions on Blockchain. In *International Conference on Information and Communication Technology Convergence, ICTC 2018, Jeju Island, Korea (South), October 17-19, 2018*. IEEE, 1204–1207. https://doi.org/10.1109/ICTC.2018.8539529

[33] Petar Kochovski, Sandi Gec, Vlado Stankovski, Marko Bajec, and Pavel D. Drobintsev. 2019. Trust management in a blockchain based fog computing platform with trustless smart oracles. *Future Gener. Comput. Syst.* 101 (2019), 747–759. https://doi.org/10.1016/j.future.2019.07.030

[34] Jae Kwon and Ethan Buchman. . *Cosmos whitepaper*. https://v1.cosmos.network/resources/whitepaper [Online, accessed 21 October 2021].

[35] Ninghui Li, John C. Mitchell, and William H. Winsborough. 2002. Design of a Role-Based Trust-Management Framework. In *2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 12-15, 2002*. IEEE Computer Society, 114–130. https://doi.org/10.1109/SECPRI.2002.1004366

[36] Ninghui Li, William H. Winsborough, and John C. Mitchell. 2003. Distributed Credential Chain Discovery in Trust Management. *J. Comput. Secur.* 11, 1 (2003), 35–86. https://doi.org/10.3233/jcs-2003-11102

[37] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 254–269. https://doi.org/10.1145/2976749.2978309

[38] Eduardo Morais, Tommy Koens, Cees Van Wijk, and Aleksei Koren. 2019. A survey on zero knowledge range proofs and applications. *SN Applied Sciences* 1, 8 (2019), 1–17.

[39] OpenZeppelin. 2021. *OpenZeppelin Access Control*. https://docs.openzeppelin.com/contracts/4.x/access-control [Online, accessed 21 October 2021].

[40] Aafaf Ouaddah, Hajar Mousannif, Anas Abou El Kalam, and Abdellah Ait Ouahman. 2017. Access control in the Internet of Things: Big challenges and new opportunities. *Comput. Networks* 112 (2017), 237–262. https://doi.org/10.1016/j.comnet.2016.11.007

[41] Juha Partala, Tri Hong Nguyen, and Susanna Pirttikangas. 2020. Non-Interactive Zero-Knowledge for Blockchain: A Survey. *IEEE Access* 8 (2020), 227945–227961. https://doi.org/10.1109/ACCESS.2020.3046025

[42] Joseph Poon and Vitalik Buterin. 2017. Plasma: Scalable autonomous smart contracts. *White paper* (2017), 1–47.

[43] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: Scalable off-chain instant payments.

[44] Raiden. 2021. *The Raiden network*. https://raiden.network/ [Online, accessed 21 October 2021].

[45] Yefeng Ruan and Arjan Durresi. 2017. A Trust Management Framework for Cloud Computing Platforms. In *31st IEEE International Conference on Advanced Information Networking and Applications, AINA 2017, Taipei, Taiwan, March 27-29, 2017*, Leonard Barolli, Makoto Takizawa, Tomoya Enokido, Hui-Huang Hsu, and Chi-Yi Lin (Eds.). IEEE Computer Society, 1146–1153. https://doi.org/10.1109/AINA.2017.108

[46] Ashish Singh and Kakali Chatterjee. 2019. Trust based access control model for securing electronic healthcare system. *J. Ambient Intell. Humaniz. Comput.* 10, 11 (2019), 4547–4565. https://doi.org/10.1007/s12652-018-1138-z

[47] Vasilios A. Siris, Pekka Nikander, Spyros Voulgaris, Nikos Fotiou, Dmitrij Lagutin, and George C. Polyzos. 2019. Interledger Approaches. *IEEE Access* 7 (2019), 89948–89966. https://doi.org/10.1109/ACCESS.2019.2926880

[48] Zhiyuan Su, Ling Liu, Mingchu Li, Xinxin Fan, and Yang Zhou. 2013. ServiceTrust: Trust Management in Service Provision Networks. In *2013 IEEE International Conference on Services Computing, Santa Clara, CA, USA, June 28 - July 3, 2013*. IEEE Computer Society, 272–279. https://doi.org/10.1109/SCC.2013.31

[49] Jason Teutsch and Christian Reitwießner. 2017. *A scalable verification solution for blockchains*. https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf [Online, accessed 21 October 2021].

[50] Yan Wang and Vijay Varadharajan. 2007. Role-based Recommendation and Trust Evaluation. In *9th IEEE International Conference on E-Commerce Technology (CEC 2007) / 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2007), 23-26 July 2007, National Center of Sciences, Tokyo, Japan*. IEEE Computer Society, 278–288. https://doi.org/10.1109/CEC-EEE.2007.83

[51] Gavin Wood. 2016. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper* (2016).

[52] Gavin Wood. 2017. Ethereum: a secure decentralised generalised transaction ledger.

[53] Zhe Yang, Kan Yang, Lei Lei, Kan Zheng, and Victor C. M. Leung. 2019. Blockchain-Based Decentralized Trust Management in Vehicular Networks. *IEEE Internet Things J.* 6, 2 (2019), 1495–1505. https://doi.org/10.1109/JIOT.2018.2836144

[54] YCharts. 2021. *Bitcoin statistics*. https://ycharts.com/indicators/sources/blockchain [Online, accessed 15 October 2021].

[55] YCharts. 2021. *Ethereum statistics*. https://ycharts.com/indicators/sources/etherscan [Online, accessed 15 October 2021].

[56] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. 2020. Solutions to Scalability of Blockchain: A Survey. *IEEE Access* 8 (2020), 16440–16455. https://doi.org/10.1109/ACCESS.2020.2967218