A Blockchain-based Framework in Support of Privacy Preferences Enforcement for Scientific Workflows

(Invited Paper)

Federico Daidone
DISTA, University of Insubria
Varese, Italy
fdaidone@uninsubria.it

Barbara Carminati
DISTA, University of Insubria
Varese, Italy
barbara.carminati@uninsubria.it

Elena Ferrari
DISTA, University of Insubria
Varese, Italy
elena.ferrari@uninsubria.it

Abstract—**Scientific workflows are today a vital tool for computational science, enabling the definition and execution of complex applications in heterogeneous and often distributed environments. A key characteristic of scientific workflow applications is that they often require the massive processing of an enormous amount of data that, in many cases, convey personal information. To allow an efficient and transparent privacy compliance check process, in this paper, we propose a blockchain-based solution coupled with an ad-hoc index structure that makes it possible an efficient compliance check for a massive amount of data.**

Index Terms—**Privacy, Scientific Workflows, Blockchain.**

I. Introduction

Today, advances in information technologies, especially those related to machine learning tools, support data-driven and information-driven science in multidisciplinary fields, such as biological, medical, physical, chemical, etc. As a result, many scientific organizations make their services available to researchers to create distributed systems capable of processing and supplying large amounts of data, allowing coordinated sharing among the parties involved, the so-called grids.

Scientists can pool these distributed resources to achieve their scientific goals, generating scientific workflows (SWFs). These resources include high performance computing services and large databases, used in pipelines. For instance, a collaborative scientific workflow can be used by organizations (e.g., research laboratories, computing centers) to jointly conduct experiments to obtain a result (e.g., particular treatments, use of new molecules or substances). In general, each piece of information that passes through the workflow has its own terms and conditions of use. More importantly, when dealing with scientific workflows, many datasets may contain highly sensitive personal information (e.g., genomics data, health status, disease, and causes of death). Some organizations may be reluctant to release and use this information without proper mechanisms to ensure controlled and secure sharing. Additionally, organizations need to be compliant with privacy regulations, such as the European General Data Protection Regulation (GDPR),[1] the California Consumer Privacy Act (CCPA),[2] or the Brazilian Lei Geral de Proteção de Dados (LGPD),[3] just to mention few.

Several proposals have successfully applied access control techniques to preserve data confidentiality in collaborative workflows (e.g., [1], [2]), but our approach differs because we adopt a user-centric vision since we deal with user privacy preferences rather than access control rules centrally managed by the security administrator. The user, aka data owner, can define his/her own privacy preferences, which represent his/her will on the processing of his/her personal data. After that, the service receives the data only if its privacy policy meets the privacy requirements of the data owner.

In this context, trust among the parties involved in the workflow is essential. Usually, a third party, in which all organizations place their trust, controls and manages both the execution of the workflow and the controlled exchange of data. However, it is not always easy to find an agreement among organizations to choose the third party that will have the control. Many scientific work have highlighted problems of trust in centralized workflows, such as data provenance and the repeatability of experiments [3], [4].

For these reasons, in this paper, we propose a distributed solution, in which the workflow is managed by the blockchain. This solution introduces numerous advantages, such as verifiability, authenticity, immutability and non-repudiation of information. By leveraging on blockchain, each party involved in the collaboration is able to monitor and audit the workflow performed on the blockchain as well as the privacy compliance process. Smart contracts, which allow the programmability of the blockchain, as in Hyperledger Fabric [5] and Ethereum [6], can be used to encode the workflow logic and checking the compliance with data owner's privacy preferences.

Performing privacy compliance during the scientific workflow execution might be time consuming due to the huge amount of data tuples to be processed, each of which might require a privacy compliance check. Therefore, to reduce the overhead of privacy preference enforcement, we propose a

[1]Available at https://gdpr.eu/

[2]Available at https://oag.ca.gov/privacy/ccpa

[3]Available at https://lgpd-brazil.info/

multi-dimensional index structure, called *IP graph*, which is built by taking into account each component of a privacy preference/policy, as well as the inclusion relationship that may hold among privacy policies/preferences. The experimental evaluation we have carried on shown the efficiency gain achieved by the proposed index structure.

Ensuring data privacy in scientific workflows has been subject to many studies (e.g., [7]–[9]), but most of the previous proposals adopt a centralized solution for privacy compliance check. There are also few more recent proposals leveraging on blockchain, but they are mainly targeting data provenance. For instance, BlockFlow [10] is a blockchain-based platform for scientific data provenance. One of the approach more related to our proposal is ProvChain [11]. Although it does not target scientific workflows, it provides a blockchain-based solution combining data provenance and privacy guarantees. However, privacy is achieved through anonymization techniques, rather than through the selective release of information driven by privacy preference enforcement. Finally, there are other work leveraging on blockchain for privacy preference enforcement, but they are targeting different application domains, such as IoT [12], mobile applications [13], or smart grids [14].

The remainder of this paper is organized as follows. Section II introduces some background information. Section III provides the general architecture of the proposed solution, while the index data structure is discussed in Section IV. Section V gives an overview of how the proposed index structure is used for privacy compliance check, whereas more details are given in Section VI. Experiments are discussed in Section VII, whereas Section VIII concludes the paper.

II. Background

A. Scientific workflows

A scientific workflow [15], [16] is a sequence of configurable activities that elaborate different datasets in order to obtain computational solutions to a scientific problem. A scientific workflow management system (SWMS) [17] manages operations, authorizations and data transfer among organizations involved in the collaboration. It provides interfaces to scientists for defining the workflow, simplifying the communication and data sharing among services, and keeping track of the data provenance (e.g., Kepler[4], or Taverna[5]). A SWMS is equipped with a workflow engine (WE) that takes care of tasks execution by properly invoking services according to the workflow order. Scientific workflows, compared to other types of workflows, such as business workflows, involve an intensive computation load and comply with the dataflow oriented model [17]. Tasks are performed by services often configured as web services (using API architectures, such as SOAP, REST, RPC, GraphQL, etc.).

Example 1: Figure 1 shows an example of scientific workflow. It represents a high level view of the process of identification and characterization of eukaryotic promoters.[6] In step 1, the microarray[7] data is selected to be analyzed using clustering techniques to find similar gene patterns (step 2). In step 3, the identifiers of the found pattern is used to search the whole sequence in a database, which is then passed to a tool that searches for similar sequences (step 4). The subsequent steps allow to generate a promoter model that is repeatedly optimized and refined until the desired result is achieved.

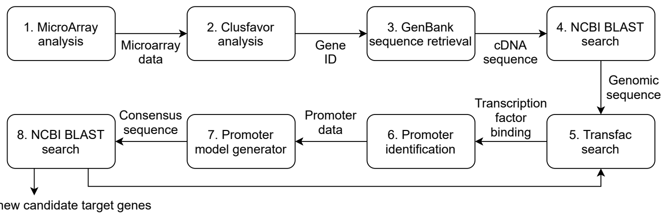


Fig. 1: An example of scientific workflow for promoter identification [17]

Example 1 shows how large amounts of data are conveyed among many actors/organizations who have different roles and functions, such as databases (e.g., GenBank) and processing tools (e.g., BLAST), that could store and process sensitive information belonging to third parties.

B. Blockchain

Blockchain is a distributed ledger technology (DLT), in which data are replicated and shared by multiple nodes [18]. In the blockchain, information is stored in blocks, cryptographically joined together to form a chain. Blocks contain transactions, recording any activity or exchange of resources made by network nodes. The consensus algorithm ensures that each node has the same data (current state of the ledger), by reaching a common agreement. Examples of consensus protocols are Proof-of-Work (PoW) and Byzantine Fault-Tolerant (BFT) [19]. By now, the vast majority of blockchains support smart contracts, that is, small autonomous programs that are executed during the transaction validation process. A blockchain can be permissionless or permissioned. Participation in permissioned blockchains is limited only to authorized and recognized nodes, whereas in permissionless ones, each node can participate in the activities of the network. Our solution uses Hyperledger Fabric,[8] a permissioned blockchain that can efficiently manage large amounts of data and supports smart contracts, called chaincodes.

C. Privacy model

We leverage on the model defined in [20] to specify privacy preferences and policies, as it is light and expressive. According to [20], purposes are structured into a purpose tree structure PT (see, Figure 2 for an example).

[4]Availabe at https://kepler-project.org/

[5]Availabe at https://incubator.apache.org/projects/taverna.html

[6]A promoter is a region of DNA, where RNA polymerase binds to initiate the transcription of one or more genes.

[7]Set of microscopic DNA spots attached to a solid surface (e.g., glass, plastic, or silicon chip) forming an array.

[8]Available at https://www.hyperledger.org/

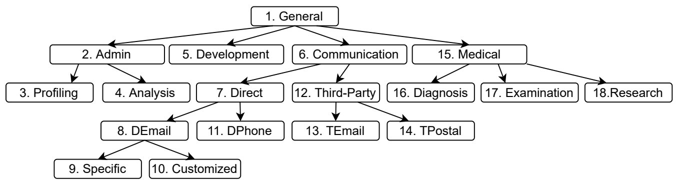


Fig. 2: An example of purpose tree

A privacy preference is formally defined as follows.

Definition 1 (Privacy preference [20]): A privacy preference is a tuple $pp = \langle \alpha, consumer, ip, rt, tpu \rangle$, where α is the data tuple to which the preference applies, $consumer$ specifies the set of consumer's identities to which pp applies, ip specifies the intended purposes for which α can be collected and used by any entity in $consumer$, rt specifies the retention time, and tpu the third party usage.[9] The intended purpose ip is in turn modelled as a pair (Aip, Exc), where Aip (allowed intended purposes) is a set of purposes belonging to a purpose tree PT and Exc (exceptions) is a set of purpose elements that descend from elements in Aip that must be denied. ip authorizes the access for all purposes that descend from the elements in Aip, except for those that descend from any element in Exc.

Example 2: Let us consider a genomic scientific workflow where several services interact to collaborate in research. A data owner wants to grant his/her data for this research but he/she wants his/her data to be used for administrative analysis purposes and all medical purposes, but not for development of services on his/her personal data. Furthermore, data can be used for 5 years, prohibiting dissemination to third parties, that is, those parties not directly involved in the workflow. These requirements, can be modeled by the following privacy preference: $pp = \langle genomicData, \{genomicResearch\}, (\{admin, medical\}, \{development\}), 5y, unshareable \rangle$.

In contrast, a privacy policy is modeled as a tuple: $\langle \alpha, up, dataRet, dataRel \rangle$, where α is the data tuple to which the policy refers to, up is the data usage purpose, $dataRet$ is the data retention time, and $dataRel$ is the third party usage. The data usage purpose $up = (Aup, Exc)$ is modelled as the intended purpose ip.

Example 3: A consumer privacy policy, stating that genomic data are collected and used for medical purposes only, they are collected for no more than 3 years and without the ability to share them with third parties, can be modeled as: $p = \langle genomicData, (\{medical\}, \{\}), 3y, unshareable \rangle$.

Therefore, to check the compliance of a privacy policy with a privacy preference we have first to check if the purposes derived from an intended purpose contains the set of purposes derived from the policy data usage purpose. Formally, we denote with $\overrightarrow{ip}$ (resp. $\overrightarrow{up}$) the set of purposes implied by ip (resp, up). $\overrightarrow{ip}$ consists of the set of elements in the purpose tree PT that descend from each $purp$ in $ip.Aip$, from which the elements that descend and ascend from each $purp$ in $ip.Exc$ have been pruned. Computation of $\overrightarrow{up}$ follows the same strategy.

During the privacy compliance check, we verify that: (1) $p.\overrightarrow{up}$ is contained in $pp.\overrightarrow{ip}$, (2) $p.dataRet$ is less than or equal to $pp.rt$ and (3) $p.dataRel$=$pp.tpu$. If these checks succeed, then the privacy enforcement is successful. In what follows, we use the notation $PE(pp, p)$ to indicate the compliance check of a privacy preference pp against a privacy policy p. The result of $PE(pp, p)$ is $true$, if p respects the constraints specified in pp, $false$, otherwise.

[9]For simplicity, in what follows, we assume that the retention time is expressed in days and the third party usage tpu assumes one of two values, namely $shareable$ or $unshareable$.

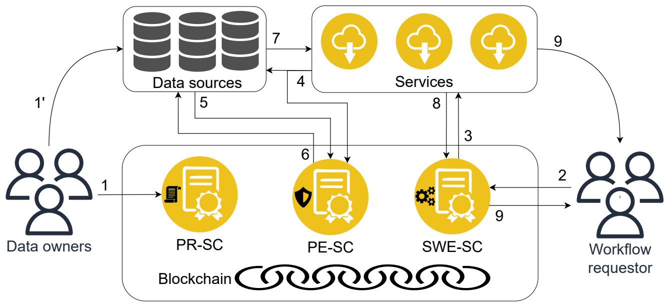


Fig. 3: Overview

III. Architecture

As depicted in Figure 3, our reference architecture implies different actors. We have the workflow requestor (WR), which is the entity (e.g., a person or a company) invoking a SWF. Referring to Example 1, a requestor could be a researcher providing an input microarray data in order to obtain new candidate target genes. Workflow tasks are executed via services delivered by different service providers (SP), such as the BLAST tool or a promoter model generator of Example 1. Services might need to process datasets managed by external data sources (DS), which handle data collected by different individuals (data owner DO).

We leverage blockchain to govern the workflow deployment, with the aim of ensuring that the data owners' privacy preferences are respected by every task execution. We assume that when a data owner uploads his/her data to a data source (see, Fig. 3, step 1'), he/she also registers the corresponding privacy preference in the blockchain. This is done through a dedicated smart contract, called PR-SC (see, Fig. 3, step 1). Privacy preferences are specified according to the model presented in the Section II-C.

Similarly to other blockchain-based workflow execution platforms (cfr. [21], [22]), we assume the presence of an off-chain entity, called *Deployer*, in charge of encoding the workflow into a smart contract, called Scientific Workflow Engine Smart Contract (SWE-SC).[10] The smart contract is then deployed on the blockchain to manage tasks/services invocation, following the defined workflow. Thus, when the

[10]The design of a smart contract for workflow management has been extensively covered by previous work, such a [21] and [22]. Therefore, we do not focus here on the *Deployer* component.

requestor wants to execute a SWF, it invokes the corresponding SWE-SC (see, Fig. 3, step 2).

During workflow execution, when a service is invoked (see, Fig. 3, step 3), the service provider generates a request to a data source for the data needed for service execution. The request contains also the service privacy policy (cfr. Section II-C) (see, Fig. 3, step 4). Once the data source has retrieved all data tuples needed by the service, it asks the blockchain to verify privacy compliance (see, Fig. 3, step 5). This request triggers the privacy enforcement that is executed via the *privacy enforcement smart contract* (PE-SC). This smart contract verifies the privacy preferences associated with requested data tuples against the service's privacy policy. Once done, PE-SC communicates to the data source the selection of tuples that can be released to the service, that is, those whose associated privacy preferences are satisfied by service's policy (see, Fig. 3, step 6). These tuples are then released to the requiring service (see, Fig. 3, step 7). In general, this release involves the exchange of large amounts of data, which is managed off-chain. This is out of scope of this paper, but it can be done following the approach in [22]. Once the service terminated its task, it returns the results to SWE-SC to proceed with the next steps (see, Fig. 3, step 8). Steps 3-8 in Figure 3 are repeated until the end of the scientific workflow. Lastly, the service releases output data to the requestor (see, Fig. 3, step 9).

Having privacy enforcement done during the workflow execution might be not feasible in the context of scientific workflows, characterized by huge amount of data tuples to be processed, each of which might require a privacy compliance check. At the task level, a way to optimize this process is to collect all tuples with the same privacy preference, such to perform only one privacy enforcement against the provider's privacy policy. Even if this represents a relevant improvement, it might not be enough to solve scalability issues. Indeed, usually multiple workflows are processed in parallel, each consisting of several tasks. Therefore, there is the need of adopting further solutions to decrease the number of privacy compliance checks.

For this purpose, we propose an alternative solution that exploits the relationships between privacy preferences. To better introduce this concept, let us consider two privacy preferences pp_1 and pp_2, whose conditions are equal except for the retention times set to 12 months for pp_1 and 6 months for pp_2, respectively. Here, we can easily see that if the provider's privacy policy satisfies the most restrictive privacy preference (i.e., the one with 6 months retention), it also satisfies the other one (i.e., the one with 12 months retention). By extending this concept to all preference's components, we introduce the *inclusion property* between two privacy preferences, formally defined as follows.[11]

Definition 2 (Inclusion property for privacy preferences): Let $pp_1 = \langle \overrightarrow{ip_1}, rt_1, tpu_1 \rangle$ and $pp_2 = \langle \overrightarrow{ip_2}, rt_2, tpu_2 \rangle$ be two privacy preferences. We say that pp_1 is included in pp_2, $pp_1 \subseteq pp_2$, if $tpu_1 = tpu_2 \wedge rt_1 \leq rt_2 \wedge \overrightarrow{ip_1} \subseteq \overrightarrow{ip_2}$.

For the sake of simplicity, Definition 2 considers only privacy preference components relevant for the inclusion (i.e., ip, rt and tpu). Conceptually, if constraints defined in a privacy preference pp are *logically contained* in those of another privacy preference pp', then pp' contains pp (denoted as $pp \subseteq pp'$) and a privacy policy p that satisfies pp also satisfies pp'. If we consider a privacy policy p and a privacy enforcement $PE(pp, p) = true$, it follows that also $PE(pp', p) = true$.

Inclusion property can be used to determine if the compliance of a policy p with a new privacy preference pp_{new} is implied by the compliance already verified with another privacy preference included in pp_{new}. An inclusion property can be defined also between two privacy policies.

Definition 3 (Inclusion property for privacy policies): Let $p_1 = \langle \overrightarrow{up_1}, dataRet_1, dataRel_1 \rangle$ and $p_2 = \langle \overrightarrow{up_2}, dataRet_2, dataRel_2 \rangle$ be two privacy policies. We say that p_1 is included in p_2, $p_1 \subseteq p_2$, if $dataRel_1 = dataRel_2 \wedge dataRet_1 \leq dataRet_2 \wedge \overrightarrow{up_1} \subseteq \overrightarrow{up_2}$.

This property can be used to further reduce the number of checks to be performed during the workflow execution. Indeed, Let us consider a privacy preference pp and two privacy policies p_1 and p_2, such that $p_1 \subseteq p_2$. Then, if p_2 satisfies pp, then also p_1 satisfies pp. Moreover, to efficiently exploit the inclusion property during workflow execution, we introduce an index structure to organize the registered privacy preferences and policies according to their inclusion properties. In particular, we propose a multidimensional index data structure tailored for blockchain and smart contracts. In what follows, we discuss the index for privacy preferences.

IV. Privacy Preferences Index

A first naive preferences index could be built by simply considering the preferences inclusion property. This could be done by creating an hierarchy on registered privacy preferences: each node corresponds to a preference, connected only with children nodes representing included privacy preferences. Even if it would bring some benefits, we can improve it by considering that preference's components are in logical conjunction (*and*). This means that we need just a not-satisfied check to conclude that the preference is not satisfied. This reduces the number of checks, avoiding unnecessary controls in case of non-compliance.

Thus, rather than exploiting an hierarchy based on the inclusion property, we propose an index structure where we exploit separately the inclusion relationship of each preference component (i.e., ip, tpu, rt). In the following, we present how the index is generated by considering each single component.

[11] For the sake of simplicity, we use the set $\overrightarrow{ip} = \overrightarrow{Aip} - \overrightarrow{Exc}$ (Sec. II-C) instead of ip, obtaining $pp = \langle \overrightarrow{ip}, rt, tpu \rangle$. The same considerations apply to the privacy policy $p = \langle \overrightarrow{up}, dataRet, dataRel \rangle$.

A. IP Graph

In this section, we focus on the privacy preference's $\overrightarrow{ip}$ component.[12] We model the inclusion relationship among $\overrightarrow{ip}$s via a directed acyclic graph (DAG), called *IP graph*. Given a set of privacy preferences, this graph is built such that any vertex represents the $\overrightarrow{ip}$ of an existing privacy preference and its edges indicate the inclusion relationships. An edge from a node n_1 to a node n_2 means that the $\overrightarrow{ip}$ corresponding to n_1 are included in the $\overrightarrow{ip}$ corresponding to n_2. Multiple pp sharing the same $\overrightarrow{ip}$ are identified by the same vertex.

Given a new pp, this is inserted in the graph only if its $\overrightarrow{ip}$ is not already present. If this is the case, we insert a new node for $\overrightarrow{ip}$. According to the *IP graph* definition, we need to add also those nodes representing the sets of purposes that could be included by $\overrightarrow{ip}$, if not already present. To compute these sets we exploit the purpose tree PT recursively as the following example clarifies.

Example 4: Consider the PT in Figure 2. For simplicity, we represent purposes with the corresponding numeric encoding shown in Figure 2.[13] Let us consider $\overrightarrow{ip_1} = \langle 2, 3, 4, 5, 8, 9, 10\rangle$. According to the PT in Figure 2, from $\overrightarrow{ip_1}$ we derive $2^{\downarrow} = \langle 2, 3, 4\rangle$, and recursively $3^{\downarrow} = \langle 3\rangle$ and $4^{\downarrow} = \langle 4\rangle$. All these vertexes are inserted in the IP graph (see Figure 4a), but we avoid connecting the latter (i.e., $3^{\downarrow}$, $4^{\downarrow}$) directly to the starting $\overrightarrow{ip_1}$ since it has been already derived from $2^{\downarrow}$. From $\overrightarrow{ip_1}$, we also get $5^{\downarrow} = \langle 5\rangle$ and $8^{\downarrow} = \langle 8, 9, 10\rangle$. Again, recursively, from $\langle 8, 9, 10\rangle$ we get $9^{\downarrow} = \langle 9\rangle$ and $10^{\downarrow} = \langle 10\rangle$. Let us now suppose that a second $\overrightarrow{ip_2} = \langle 2, 3, 4, 7, 8, 9, 10, 11\rangle$ is inserted. In this case, we connect the existing nodes (i.e., $\langle 2, 3, 4\rangle$ and $\langle 8, 9, 10\rangle$) and insert only the new ones $7^{\downarrow} = \langle 7, 8, 9, 10, 11\rangle$ and $11^{\downarrow} = \langle 11\rangle$.

As the example shows, the index requires the creation of ip nodes for which there is no privacy preference among those registered (e.g., nodes $\langle 2, 3, 4\rangle$ in Figure 4a), but this simplifies and speeds up the enforcement. Hereafter, we refer to nodes for which there is a privacy preference as *real nodes*, denoted as rectangles in Figure 4a, while we refer to the others as *link nodes*, represented by an oval/circle.

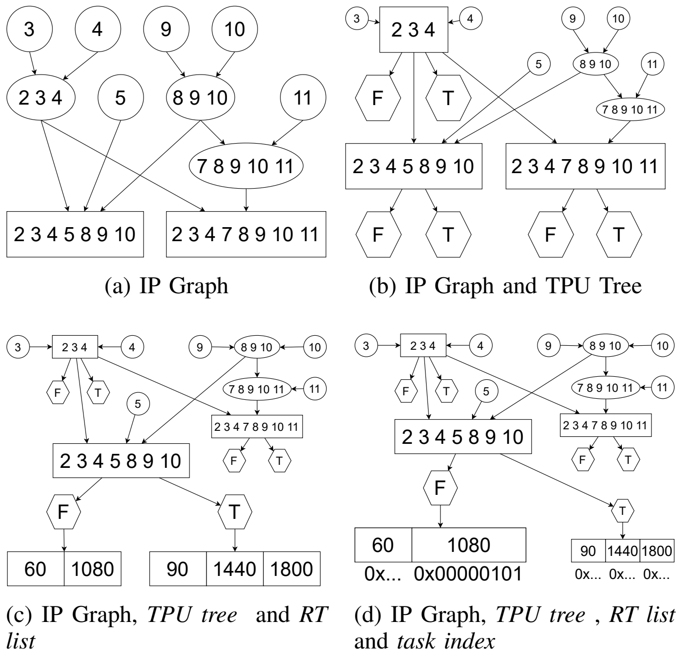
B. TPU tree

In the *IP Graph*, we have that all pps sharing the same $\overrightarrow{ip}$, i.e. $\langle\overrightarrow{ip}, *, *\rangle$,[14] are associated with the same node. By considering the tpu component, we index all the pp that also have the same tpu, i.e. $\langle\overrightarrow{ip}, *, tpu\rangle$. The tpu component can only have two values, $true$ or $false$. Therefore, we can represent it through a 1-level tree, rooted on a *real node* $\overrightarrow{ip}$ and with $false$ and $true$ as leaves. We call this structure *TPU tree*.

[12]For sake of simplicity, we refer to ip as its implied intended purpose set $\overrightarrow{ip}$.

[13]We encode PT with a pre-order numbering, starting from the root as value 1 and using increasing integers for every other purpose. For instance, $\{Admin, Profiling, Analysis\}$ is represented by $\{2, 3, 4\}$.

[14]We use the wildcard character "$*$" to indicate any value of the corresponding component.



(a) IP Graph

(b) IP Graph and TPU Tree

(c) IP Graph, *TPU tree* and *RT list*

(d) IP Graph, *TPU tree* , *RT list* and *task index*

Fig. 4: Example of index data structures

Example 5: Let us assume to consider two new pps: $pp_1 = \langle\langle 2, 3, 4\rangle, 270, false\rangle$, and $pp_2 = \langle\langle 2, 3, 4\rangle, 180, true\rangle$. These two proferences have the same ip, that is, $\langle 2, 3, 4\rangle$, but two different tpu values. As a result, as represented in Figure 4b, $\langle 2, 3, 4\rangle$ will become a *real node*, and it can be used to index $pp_f = \langle\langle 2, 3, 4\rangle, *, false\rangle$ or $pp_t = \langle\langle 2, 3, 4\rangle, *, true\rangle$ in the *TPU tree*, represented with hexagons.

C. RT list

The rt component represents the retention time expressed in days. In the enforcement phase, it must be easy to select a subset of rt greater than or equal to a given value. At this aim, we exploit a list, for storing an ascending ordered sequence of rt values.

More precisely, for each *ip node*, we have two *RT lists*: one stored in the "F" leaf (for $tpu = false$), and one stored in "T" (for $tpu = true$). In both the cases, elements in *RT lists* contain for each rt value the list of associated privacy preferences. These represent the privacy preferences having the same ip, tpu and rt values. The ids of these preferences are contained into the $idPPs$ element attached to each rt element.

Example 6: Let us assume that two new pp have been considered: $pp_1 = \langle\langle 2, 3, 4, 5, 8, 9, 10\rangle, 60, false\rangle$ and $pp_2 = \langle\langle 2, 3, 4, 5, 8, 9, 10\rangle, 1080, false\rangle$. Both pp have common $\langle\langle 2, 3, 4, 5, 8, 9, 10\rangle, *, false\rangle$, so we can use nodes $ip = \langle 2, 3, 4, 5, 8, 9, 10\rangle$ and $tpu = false$ to store them. The connected *RT list* (represented by sequences of rt values) has to hold $rt = \{60, 1080\}$. The list for $tpu = true$, instead, hosts three new pp with $rt = \{90, 1440, 1800\}$.

D. Task index

In order to support privacy enforcement for multiple tasks simultaneously, the proposed index also associates with each registered pp (i.e., to its components) the list of tasks where this preference is used. This list contains those tasks that require at least a data tuple to which pp applies. At this aim, each task is assigned a $indexTask$, set as a counter starting from 1. The list of tasks is encoded as a bit-string, where the j-th bit corresponds to task with $indexTask = j$ and it is set to 1 if task with $indexTask = j$ is associated with pp. The list of tasks where a pp is used, aka $taskIDs$ bit-string, is thus associated with the *RT list* element representing that pp, as depicted in Figure 4d.

V. Privacy Enforcement

In this section, we show how the proposed index is used to perform privacy preferences' enforcement. In particular, we first present the enforcement for a single task t. Then, we discuss how the enforcement is extended to support multiple tasks.

A. Single task privacy enforcement

Let us consider a task t, with $indexTask_t$, that has to be executed by a service provider SP_t. In order to proceed with task execution, SP_t requests to a data source DS the access to a set of tuples and submits its privacy policy to the blockchain. As illustrated in Section III, before releasing the requested data, DS triggers the privacy enforcement. At this purpose, the blockchain retrieves all privacy preferences applied to data tuples required by t and generates the *privacy preference index*, say *IP graph*$_t$, as described in Section IV. To evaluate all these preferences against the privacy policy p of provider SP_t, we exploit *IP graph*$_t$. In particular, we can see p as an entry point in *IP graph*$_t$, by locating the node in *IP graph*$_t$ modeling the ip component with value equal to $p.\overrightarrow{up}$.

Let n be such an entry node.[15] As a first step, we retrieve the set of real nodes RN contained in the subtree rooted at n. As an example, considering the *IP graph* in Figure 4d and assuming the link node $\langle 8, 9, 10\rangle$ as the entry node for p, then real nodes to be evaluated are: $\langle 2, 3, 4, 5, 8, 9, 10\rangle$ and $\langle 2, 3, 4, 7, 8, 9, 10, 11\rangle$. Then, for each $rn \in RN$, we check its *TPU tree* and *RT list*. In particular, we follow the *TPU tree* branch based on $p.dataRet$. Then, we select in the retrieved *RT list* all rt elements greater than or equal to $p.dataRel$. Let refer to this set as rt_sat. For each $rt \in rt\_sat$, we check if $indexTask_t$ is present in its $taskIDs$. If this is the case, p satisfies all privacy preferences modelled by that rt element. Thus, SP_t is authorized to receive tuples associated with these privacy preferences. Therefore, DS will send SP_t only those tuples required by $indexTask_t$ to which one of the satisfied privacy preferences apply.

Example 7: Let us consider the example shown in Figure 4d and suppose that the policy $p = \langle\langle 8, 9, 10\rangle, false, 720\rangle$ is associated with the service provider requiring to execute task with $taskIndex = 1$. The link node $\langle 8, 9, 10\rangle$ is the entry point for p. Starting from it, we reach the first real node $\langle 2, 3, 4, 5, 8, 9, 10\rangle$, with $tpu = false$. The rt list returns $\{60, 1080\}$. Considering $dataRet = 720$, we select $rt = 1080$ because it is greater than $dataRet$. Thus, policy p satisfies the privacy preferences associated with this rt element, that is, those whose ids are contained in the corresponding $IdPPs$. Moreover, this rt element has an $idTask = 0x0000101$, indicating that $taskIndex = 1$ requires at least a data tuple to which are applied a preferences in $IdPPs$. This implies that the service provider has to receive those tuples required by $taskIndex = 1$ to which a preferences in $IdPPs$ apply. This process is repeated to analyze the other real nodes and compute the final set of tuples to be returned.

[15]If in does not exist, we insert a new *link node* in *IP graph*$_t$ with value $p.\overrightarrow{up}$ value, and we proceed as described.

B. Multiple tasks privacy enforcement

The enforcement described in the previous section can be optimized to simultaneously support multiple tasks' execution. In particular, we assume that the system has a list of tasks $\mathcal{T}$ to be performed simultaneously and each task t has its own privacy policy p, forming the policy set $\mathcal{P}$. Then, rather than generating a different index for each task in $\mathcal{T}$, we exploit the policy inclusion property to reduce the number of indexes. In particular, we order all policies in $\mathcal{P}$ in a queue $q = \{p_1, \ldots, p_n\}$, where for each pair p_j and p_{j+1} it holds that $p_{j+1} \subseteq p_j$. Thus, if we associate a unique *privacy preference index*, hereafter *IP graph*$_q$, to q (aka to the corresponding tasks), we can reduce the number of compliance checks. Indeed, if the more restrictive policy satisfies a privacy preference(s) in *IP graph*$_q$, then all the more permissive ones satisfy it as well. The generation of q and *IP graph*$_q$ is done iteratively by considering each task t in $\mathcal{T}$. More precisely, for each task t, we check if its p can be inserted in the existing q. If so, we insert p in queue q and update *IP graph*$_q$ by including the privacy preferences associated with data tuples considered by t. If p cannot be inserted in q, that is, it does not satisfy the inclusion property for any of the policies in q, then a new queue q' is created with a new associated *privacy preference index*, *IP graph*$'_q$. At the end, the blockchain maintains the set of created queues $\mathcal{Q}$, with the corresponding *IP graph*$_q$, $\forall q \in \mathcal{Q}$. For the sake of clarity, in the following, we explain privacy enforcement by assuming the presence of a unique q ordering all policies associated with $\mathcal{T}$.

We start by considering the most restrictive policy in q, i.e., p_1, and by following the single task enforcement procedure. Thus, we search for the entry node n in *IP graph*$_q$ referring to p_1, that is, the node with value equal to $p_1.\overrightarrow{up}$. Similarly to the single task case, when the entry point does not exist, a link node with value $p_1.\overrightarrow{up}$ is inserted. Then, we retrieve the set of real nodes RN contained in the subtree rooted at n. Given $rn \in RN$, we consider its *TPU tree* branch based on $p_1.dataRet$. Thus, we select in the retrieved *RT list* all rt greater than or equal to $p_1.dataRet$. This represents all privacy preferences satisfied by p_1. According to the definition of privacy preference indexes, each rt in *RT list* has associated the $idTask$ element storing the tasks consuming data tuples which are covered by preferences indexed by rt. Hereafter, we denote as $AuthTask_{p_1}$ ($SatPref_{p_1}$ resp.) the union of task's ids (privacy preference's ids, resp.) contained in $idTask$ elements ($idPPs$ elements, resp.) of the retrieved rt (those with rt greater than or equal to $p_1.dataRet$). Thus, all privacy preferences $SatPref_{p_1}$ applying to at least a data tuple consumed by tasks in $AuthTask_{p_1}$ are satisfied by p_1. Since p_1 is the most restrictive policy, then all other policies in q satisfy $SatPref_{p_1}$ as well. This implies that all tasks corresponding to policies in q are authorized to receive some tuples. More precisely, each task receives only those required tuples to which is applied a preference in $SatPref_{p_1}$.

Similarly to single task enforcement procedure, the above process is repeated for each $rn \in RN$. Note that all retrieved rt elements are marked, so to avoid to re-consider them in the next evaluation. Once all rn has been evaluated, we re-start the enforcement by processing the second policy, p_2, in the queue, and so forth till all policies in q have been evaluated. It is important to note that, at each iteration, the set of nodes in *IP graph*$_q$ to be evaluated is less than the previous one, as we avoid to re-evaluate privacy preferences (i.e., rt elements) already satisfied by a more restrictive policy.

VI. Blockchain-based workflow execution

As a first step (step 1, Fig.3), the data owner creates and sends his/her privacy preference *pp* to the blockchain, using a privacy preference tuple defined as $t_{pp} = \langle idTpp, idDO, idS, pp\rangle$, where *idTpp* is the privacy preference tuple identifier, *idDO* is the data owner identifier, *idS* is the service identifier (aka *consumer* in the privacy model presented in Sec. II-C) to which privacy preference *pp* applies. Through a smart contract, the link between a tuple, identified by *idTpp*, and the associated *pp* is saved on-chain as $(idTpp, t_{pp})$. In parallel, the data owner sends his/her data to the data sources (step 1', Fig.3), using a data tuple $t_d = \langle idTd, idDO, idTpp, d\rangle$, where *idTd* is the data tuple identifier, *idDO* is the data owner identifier, *idTpp* is the privacy preference identifier, and d are the data on which *idTpp* applies.

The requestor starts a new workflow via the smart contract SWE-SC, implementing the workflow engine (step 2, Fig.3).[16] For each task t of the workflow, SWE-SC contacts the target service by sending the instructions necessary to perform the corresponding job (step 3, Fig.3). As described in step 4 (Fig.3), the target service might need to access some data stored in a data source to perform its task. If this is the case, task t is inserted into the list of tasks $\mathcal{T}$.

Privacy enforcement is handled by the *Privacy enforcement smart contract* (PE-SC) (see Pseudocode 1). This smart contract is invoked by a trigger stored in the blockchain. The trigger can be set on a time basis, e.g., every 500 milliseconds, or based on the number of tasks in $\mathcal{T}$, e.g., every 10 tasks.

[16]We do not describe this smart contract. Interested readers can find more details in [22], [23].

Pseudocode 1: Privacy enforcement smart contract (PE-SC)

```
Data: $\mathcal{T}$, the set of tasks to be executed
1  Function MultipleTasksEnf ($\mathcal{T}$)
2      Let $\mathcal{T}$ be the set of tasks to be executed;
3      Let $\mathcal{P}$ be the set of policies associated with tasks in $\mathcal{T}$;
4      $\mathcal{Q} = generateQs(\mathcal{P})$;
5      forall $q \in \mathcal{Q}$ do
6          $IPgraph_q = generateIPgraph(q)$;
7          while $q! = \{\}$ do
8              Let $AuthTuples$ be initialized empty;
9              Let $p_1$ be the first policy in $q$;
10             ($AuthTasks_{p_1}$, $SatPref$)= $SingleEnf(p_1, IPgraph_q)$;
11             Let $IdsTuples_{AuthTasks}$ be the ids of tuples required by tasks in $AuthTasks_{p_1}$;
12             Let $IdsTuples_{SatPref}$ be the ids of tuples to which preferences in $SatPref$ apply;
13             $AuthTuples$= $IdsTuples_{AuthTasks} \cap IdsTuples_{SatPref}$;
14             Return: $AuthTuples$ to all tasks in $\mathcal{T}$ with policies in $q$;
15             $q = q \setminus p_1$;
16         end
17     end
18 end
19 Function SingleEnf ($p$,$IPgraph$)
20     Let $p$ be a privacy policy;
21     Let $IPGraph$ be a privacy preferences index;
22     Let $n$ be the node with value equal to $p.\overrightarrow{up}$;
23     Let $RN$ be the set of real nodes in the subtree rooted at $n$;
24     Let $AuthTasks$ and $SatPref$ be initialized empty;
25     forall $rn \in RN$ do
26         Let $rn$.TPU tree be the TPU branch with value equal to $p.dataRet$;
27         Let $rt$ be the set of RT list in $rn$.TPU tree;
28         $rt_{sat} = \{rt' \subset rt | rt' \geq p.dataRel\}$;
29         forall $\overline{rt} \in rt_{sat}$ do
30             $AuthTasks = AuthTasks \bigcup \overline{rt}.idTask$;
31             $SatPref = SatPref \bigcup \overline{rt}.idPPs$;
32         end
33     end
34     Return: $AuthTasks$, $SatPref$;
35 end
```

In particular, the trigger invokes the $MultipleTasksEnf$ function in PE-SC by passing $\mathcal{T}$ as input. As described in Section V-B, given the privacy policies associated with tasks in $\mathcal{T}$, the function computes all the corresponding queues and order them (line 4). Each queue q is then considered (loop in line 5) by generating a distinct *IP graph*$_q$ on which each policy in q is evaluated (loop in line 7). This loop starts from the most restrictive policy in q (i.e., p_1), by then removing it from q before the new iteration (line 15). To evaluate a given p on *IP graph*$_q$, we exploit function $SingleEnf$ (line 10). This function implements the single enforcement process described in Section V-A. In particular, this function: searches for the entry node n in *IP graph*$_q$ with value equal to $p.\overrightarrow{up}$ (line 22); retrieves the set of real nodes RN rooted at n (line 23); considers only their *TPU tree* branches satisfying $p.dataRet$ (line 26); computes their rt_{sat} elements having values greater than or equal to $p.dataRel$ (line 28). This set of rt elements represents all privacy preferences satisfied by policy p. Then, $SingleEnf$ function returns to $MultipleTasksEnf$ the list of satisfied privacy preference $SatPref$ (line 31) and the list of authorized tasks $AuthTasks$ (line 30). To compute the tuples to be released, the $MultipleTasksEnf$ retrieves the set of tuples required by tasks in $AuthTasks$ and the set of tuples to which a privacy in $SatPref$ is applied (lines 11 and 12). The intersection between these two sets consists of the tuples to be released to all tasks whose policies are in q (line 13) Indeed, since the considered policy is the most restrictive in q, then all other policies satisfy $SatPref$ as well.

VII. Experimental results

We run a set of experiments to test the efficiency of the proposed solution. We leverage on a dedicated Ubuntu server 18.04.5 LTS, with AMD Ryzen Threadripper 1950X 16-Core processor, 32GiB system memory, and 256GB NVMe disk storage. We use Hyperledger Fabric v2.2 LTS with Docker container to run peers.

In what follows, we first introduce the used datasets and then the obtained results.

A. Datasets

For our tests we used a synthesized dataset, since we are not aware of a real publicly available dataset containing both privacy policies and preferences.

The *number of tasks* is an essential parameter to test the overhead of our privacy enforcement. A task is equivalent to a certain number of privacy preferences' compliance checks. However, a privacy preference can cover multiple data tuples, because the same privacy preference can be associated with different data tuples. We use *coverage* to measure the relationship between the amount of privacy preferences and the total number of data tuples of a task. We assume that the coverage is uniformly distributed among privacy preferences, for example a coverage of 5% means that a privacy preference covers an average of 20 data tuples out of 100 of a given task. Therefore, if a task has 100,000 data tuples with coverage = 1%, it means that there are 1,000 tuples to which apply the same privacy preference. Our system supports multi-task scenarios, that is, privacy enforcement for different tasks within a single execution of the corresponding smart contract. In this case, we set the same coverage for each task, so as to have results with the same amount of privacy compliance checks

The other two parameters are the *complexity* of the privacy policies/preferences and their *selectivity*. The complexity of a privacy policy (resp. privacy preference) is determined by the operations necessary to carry out the privacy enforcement through the proposed algorithm. For privacy preferences, the greater the number of purposes $purp[\#]$ in ip, a greater number of nodes are created in the IP graph. For tpu, choosing $true$ or $false$ is irrelevant to the computation. In contrast, a large rt value, $rt[d]$, means a longer traversal time for the rt list. Thus, we define three complexity classes:

- Low: $1 \leq purp[\#] \leq 6 \wedge 1 \leq rt[d] \leq 1,200$;
- Mid: $7 \leq purp[\#] \leq 12 \wedge 1,201 \leq rt[d] \leq 2,400$;
- High: $13 \leq purp[\#] \leq 18 \wedge 2,401 \leq rt[d] \leq 3,600$.

Assuming an index with privacy preferences uniformly distributed between low, mid and high complexity, we note that the lower the number of purposes in up is, the greater is the complexity because there are more nodes in the $IPgraph$ to be analyzed; the smaller the $dataRet$ value, $dataRet[d]$, is, the greater the complexity is because the values to be analyzed in the $RTlist$ are more; instead the complexity does not vary as the $dataRel$ varies as it is a single value. Also in this case we define three complexity classes for privacy policies:

- Low: $13 \leq purp[\#] \leq 18 \wedge 2,401 \leq dataRet[d] \leq 3,600$;
- Mid: $7 \leq purp[\#] \leq 12 \wedge 1,201 \leq dataRet[d] \leq 2,400$;
- High: $1 \leq purp[\#] \leq 6 \wedge 1 \leq dataRet[d] \leq 1,200$.

Finally, selectivity is the ratio between the privacy preferences that pass the compliance check and the total number of privacy preferences.

To evaluate our framework under different scenarios, we generated several datasets under the following conditions: Privacy policy complexity: Low, Mid, $High$; $coverage$: 1%, 5%, 10%; $selectivity$: 50%, 75%, 100%; $tasks$: 1-10.

Privacy preferences that populate the datasets have complexity uniformly distributed between low, mid, and $high$. In total we generated 270 datasets and ran each test 10 times to get more accurate results. The values reported in what follows are the average values of the 10 executions. By sizing the data reply to 100,000 data tuples, as the coverage varies we get 1,000 (1%), 5,000 (5%), 10,000 (10%) privacy preference per task. In the case of coverage at 10% and 10 tasks, we reached 100,000 privacy compliance checks in a single execution of the smart contract.

B. Results

To test the performance gain of our proposal, we compare it against a naive implementation of the compliance checks, where instead of creating a dedicated index, privacy preferences are stored into lists, one for each task.

All combinations of tests described in the previous section were carried out, producing a large amount of results. To present them, we start by setting coverage at 5% and complexity at Mid value. This can be considered as an average case, reflecting a real system load, where heterogeneous privacy preferences limit the grouping capacity of the *IP graph*, as they contain different ip, tpu and rt.

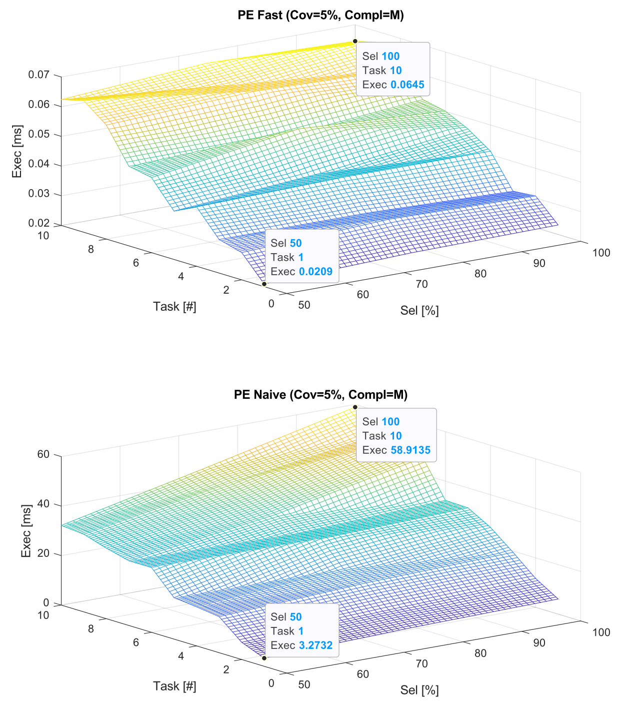


Fig. 5: Privacy enforcement execution time of our approach (fast) compared with the naive approach

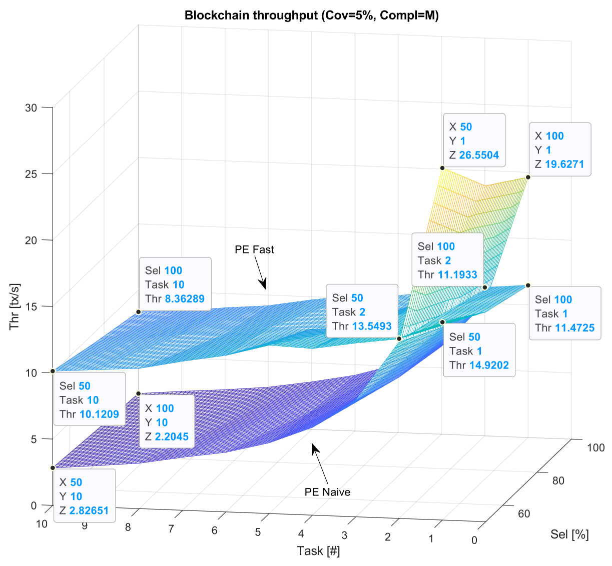


Fig. 6: Blockchain throughput measured in transactions per second

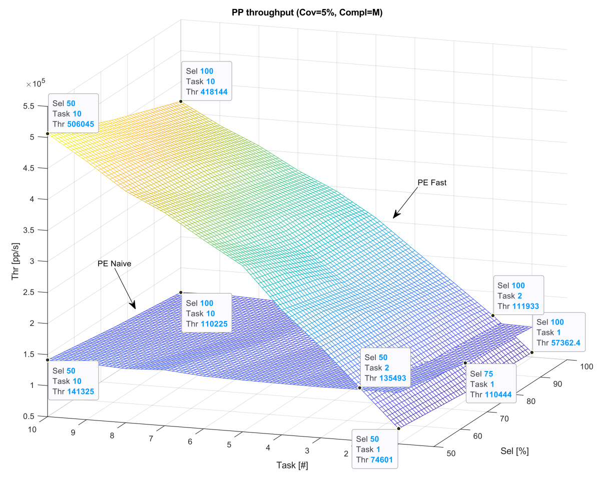


Fig. 7: Privacy preference throughput

The privacy enforcement execution time is the net time required to execute privacy enforcement for all the tasks in the input queues $\mathcal{Q}$. This measure does not take into account the time needed by the blockchain to reach the distributed consensus. Figure 5 shows the trend, fixing complexity and coverage as stated before, and varying the number of tasks and selectivity. At maximum load, with our approach we spend 0.0645 ms, whereas the naive implementation takes 58,9135 ms, and this trend increases as the number of tasks and selectivity increase.

We also measure the transaction throughput, that is, the number of blockchain transactions performed in one second. Figure 6 shows the throughput trend of our approach compared to the naive one. The naive algorithm is more efficient on a single task. In contrast, its performance decreases quickly when dealing with multiple tasks, reaching 19-26 tx/s. In contrast, our approach, which has been designed to perform privacy enforcement of multiple tasks, maintains a linear trend as the number of tasks and selectivity increase, until 8-10 tx/s.

Finally, the privacy preference throughput shows the number of privacy preferences processed per transaction in one second. As reported in Figure 7, for our approach, we obtain higher throughput values then the naive approach, starting from 2 tasks, reaching peaks of 500,000 privacy preferences per second, whereas in the naive case, the trend is around 110,000-141,000 privacy preferences per second.

Finally, to avoid showing all the graphs, for each considered combination of coverage and complexity values, we summarize the privacy enforcement execution time in a single graph. Each point in Figure 8 represents the average time for all tasks (from 1 to 10) and for all selectivity (50%, 75%, 100%) with the same coverage and complexity. For instance, the value of the point corresponding to coverage 5% and Mid complexity (M = 2), represents the average of all times reported in the graph in Figure 5. This graph does not give absolute information on the execution time, rather it shows how the trend evolves while increasing complexity and coverage.

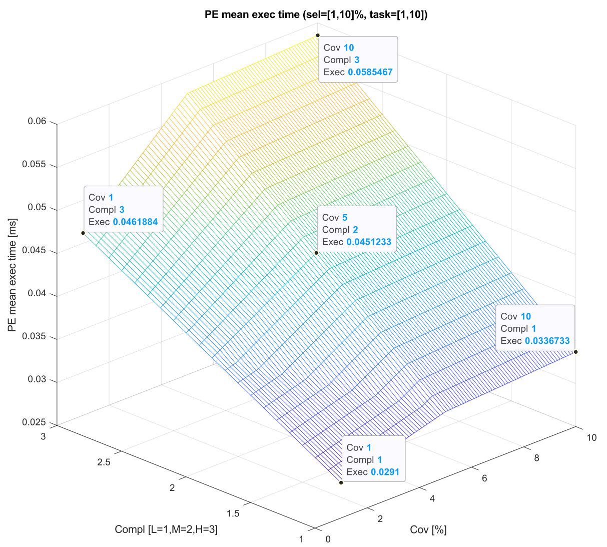


Fig. 8: Privacy enforcement mean execution time when coverage varies from 1% to 10% and complexity from L to H

VIII. Conclusion

In this paper, we presented a privacy-preserving blockchain-based scientific workflow platform, able to perform privacy enforcement in an efficient and scalable way. This is achieved through a multi-dimensional index structure that speeds up the process of privacy compliance check. We plan to extend the proposed solution along several directions. First, we plan to integrate a greater number of features in the supported privacy model (e.g., automatic privacy preference generation for newly generated information , etc.). We also plan to study techniques to further improve the performance and scalability of our system.

Acknowledgment

This work has received funding from CONCORDIA, the Cybersecurity Competence Network supported by the European Union's Horizon 2020 Research and Innovation program under grant agreement No 830927.

References

[1] A. Chebotko, S. Lu, S. Chang, F. Fotouhi, and P. Yang, "Secure abstraction views for scientific workflow provenance querying," *IEEE Transactions on Services Computing*, vol. 3, no. 4, pp. 322–337, 2010.

[2] F. A. Bhuyan, S. Lu, R. Reynolds, J. Zhang, and I. Ahmed, "A security framework for scientific workflow provenance access control policies," *IEEE Transactions on Services Computing*, 2019.

[3] D. Fernando, S. Kulshrestha, J. D. Herath, N. Mahadik, Y. Ma, C. Bai, P. Yang, G. Yan, and S. Lu, "Sciblock: A blockchain-based tamper-proof non-repudiable storage for scientific workflow provenance," in *2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2019, pp. 81–90.

[4] W. Chen, X. Liang, J. Li, H. Qin, Y. Mu, and J. Wang, "Blockchain based provenance sharing of scientific workflows," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 3814–3820.

[5] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 30.

[6] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger. ethereum project yellow paper 151 (2014)," 2014.

[7] Y. Gil, W. K. Cheung, V. Ratnakar, K. kin Chan, T. Finin, L. Kagal, and D. Olmedilla, "Privacy enforcement in data analysis workflows," 2007.

[8] K. Belhajjame, N. Faci, Z. Maamar, V. A. Burégio, E. Soares, and M. Barhamgi, "Privacy-preserving data analysis workflows for escience." in *EDBT/ICDT Workshops*, vol. 2322, 2019.

[9] S. Besik and J.-C. Freytag, "Ontology-based privacy compliance checking for clinical workflows," 09 2019.

[10] R. Coelho, R. Braga, J. M. David, F. Campos, and V. Ströele, "Blockflow: Trust in scientific provenance data," in *Anais do XIII Brazilian e-Science Workshop*. SBC, 2019.

[11] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 468–477.

[12] F. Daidone, B. Carminati, and E. Ferrari, "Blockchain-based privacy enforcement in the iot domain," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[13] J. O. Umekwudo and J. Shim, "Blockchain technology for mobile applications recommendation systems," *The Journal of Society for e-Business Studies*, vol. 24, no. 3, pp. 129–142, 2019.

[14] I. Sestrem Ochôa, L. Augusto Silva, G. De Mello, N. M. Garcia, J. F. de Paz Santana, and V. R. Quietinho Leithardt, "A cost analysis of implementing a blockchain architecture in a smart grid scenario using sidechains," *Sensors*, vol. 20, no. 3, p. 843, 2020.

[15] A. Barker and J. Van Hemert, "Scientific workflow: a survey and research directions," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2007, pp. 746–753.

[16] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *2008 third workshop on workflows in support of large-scale science*. IEEE, 2008, pp. 1–10.

[17] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and computation: Practice and experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[18] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Business & Information Systems Engineering*, 2017.

[19] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International workshop on open problems in network security*, 2015.

[20] G. Sagirlar, B. Carminati, and E. Ferrari, "Decentralizing privacy enforcement for internet of things smart objects," *Computer Networks*, vol. 143, pp. 112 – 125, 2018.

[21] B. Carminati, C. Rondanini, and E. Ferrari, "Confidential business process execution on blockchain," in *2018 ieee international conference on web services (icws)*. IEEE, 2018, pp. 58–65.

[22] C. Rondanini, B. Carminati, F. Daidone, and E. Ferrari, "Blockchain-based controlled information sharing in inter-organizational workflows," in *2020 IEEE International Conference on Services Computing (SCC)*, 2020, pp. 378–385.

[23] B. Carminati, E. Ferrari, and C. Rondanini, "Blockchain as a platform for secure inter-organizational business processes," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, 2018, pp. 122–129.