Acta Cybernetica **25** (2021) 485–516.

Evaluation of EHR Access Control in a Heterogenous Test Environment*

Zoltán Szabó[ab] and Vilmos Bilicki[ac]

Abstract

Since the advent of smartphones, IoT and cloud computing, we have seen an industry-wide demand to integrate different healthcare applications with each other and with the cloud, connecting multiple institutions or even countries. But despite these trends, the domain of access control and security of sensitive healthcare data still raises a serious challenge for multiple developers and lacks the necessary definitions to create a general security framework that addresses these issues. Taking into account newer, more special cases, such as the popular heterogeneous infrastructures with a combination of public and private clouds, fog computing, Internet of Things, the area has become evermore complicated. In this paper we will introduce a categorization of the required policies, describe an infrastructure as a possible solution to these security challenges, and then evaluate it with a set of policies based on real-world requirements.

Keywords: EHR, FHIR, telemedicine, cloud, access control, security

1 Introduction

In the mid-2010s with the emergence of the Fast Healthcare Interoperability Resources (FHIR) standard from HL7 [1], it seemed that we finally had the necessary tools to create e-health applications and databases that not only meet their respective institutional requirements, but also conform to international standards, making a networked health infrastructure more feasible. FHIR achieved this by defining a set of over 90 document templates that can be implemented in both JSON and XML formats and used to describe the entire healthcare workflow from administration to the daily events that a general practitioner or nurse is confronted

*This research was supported by the EU-funded Hungarian grants EFOP-3.6.1-16-2016-00008, EFOP-3.6.3-VEKOP-16-2017-00002, 2018-1.1.1-MKI-2018-00249 and GINOP-2.2.1-15-2017-00073

[a]Department of Software Engineering, University of Szeged, Hungary
[b]E-mail: szaboz@inf.u-szeged.hu, ORCID: 0000-0003-3863-7595
[c]E-mail: bilickiv@inf.u-szeged.hu, ORCID: 0000-0002-7793-2661

DOI: 10.14232/actacyb.290283

with. FHIR has also made these documents customizable to meet specific requirements and cover specific areas. These attractive aspects have made FHIR the most popular and widely used healthcare communications standard from HL7 to date.

However, the FHIR standard had some alarming shortcomings [7, 10]. Although some of these have been addressed in the course of the various updates to the standard, one of the most pressing is still an open problem - namely the lack of clearly defined access control and security. While FHIR generally accepts custom extensions and adaptations of its standardized document types, it provides only a light template and some minor guidelines for security policy enforcement. This has led to a "free-for-all" problem in the development of e-Health applications, with almost everyone developing their own solutions, which greatly corrupts the original concept of interoperability. With the introduction of the GDPR [12], the increasing integration of IoT and intelligent devices into the healthcare workflow [18] and in some cases the decision to use a heterogeneous backend [32] for handling accessibility and sensitive data, the complexity of this issue has increased. Another complicating factor is that the most popular current technologies for the backend are serverless and native cloud infrastructures. These present two main problems: with the advent of fog/edge computing, data processing takes place much closer to the end devices and user locations, and in these cases a large amount of sensitive data is stored in a public cloud.

The two main approaches recommended by the official documentation of the FHIR standard for these challenges are the attribute- or role-based policy controls, ABAC [36] and RBAC [15], respectively. With RBAC, the developer is able to assign specific roles to users that determine the level of access and possible operations in the system. Some typical roles in a medical system would be those of a doctor, nurse, patient, family member, and so on. In contrast, ABAC uses specific attributes of the user or the requested data to determine whether access should be granted.

However, in the current network topology, which combines IoT, intelligent devices, edge computing, private and public clouds, these methods in themselves are far from sufficient. To meet these needs, it is essential to develop a hybrid approach that combines the strengths of these two classical methods. Furthermore, it is important that these enforcement points can be placed at any part of the infrastructure to deal with the sensitive nature and processing requirements of the data. For example, while fog endpoints require a complete FHIR object, the connecting IoT devices may not be able to handle such complex data structures. In the case of a hybrid cloud solution, the data could also pass through a public cloud between the private cloud and end users, where naturally stricter policies and encryption are required than for the isolated, private parts of the infrastructure, as shown in Figure 2.

A popular concept for such enforcement points is the concept of Policy Enforcement Point (PEP), developed by the standards organization OASIS as part of its eXtensible Access Control Markup Language standard [14], an extension of the classic ABAC model, also known as policy-based access control or PBAC. While we are committed to developing a custom, fine-grained security solution that is

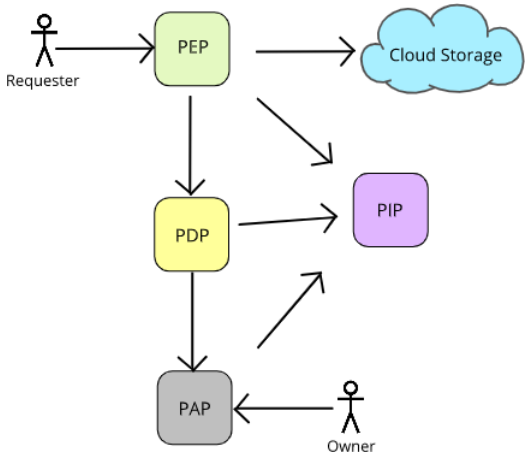

Figure 1: The Policy Enforcement Flow between end users and the cloud

not subject to the strict limitations of the XACML standard, the concept of the PEP-based architecture is well suited to our needs. In the complete model shown in Figure 1, the responsibilities for access control and security enforcement are distributed across several components. The Policy Enforcement Point (PEP) is the key to the model that enforces policies and allows or denies access to resources. Administrators can define given policies at the Policy Administration Point (PAP), which are evaluated and stored by the Policy Decision Point, based on the user's identity or multiple identities, and recognized by the Policy Identification Point (PIP). The PDP's decision is handled and enforced by a PEP. This model also provides some room for customization, because the exact structure of these nodes can be defined by the developers, and the nodes have the ability to fuse multiple elements of the infrastructure into one.

The main requirements of such a PEP solution in our infrastructure are the following:

- **Transparency**: It should have as little impact as possible on the performance and latency of the system;
- **Efficiency**: Since several elements of the infrastructure do not have the memory and CPU capacity to perform complex transformations and an analysis of the data, the enforcement engine should spare them from the more demanding operations;
- **Portability**: It should be possible to place it at any point on the infrastructure. The main strength of edge computing is its ability to provide functionality even when the cloud is not available. This also means that it should be able to work between the edge and the cloud, between the edge and end-

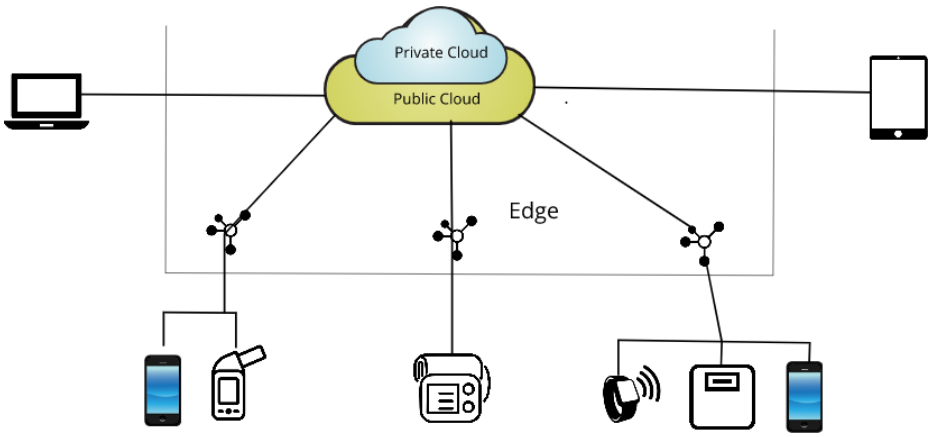

Figure 2: A hybrid edge computing infrastructure with a public and a private cloud

points, in the cloud or in some scenarios even on endpoints if they have the necessary resources;

- **Adaptability**: The domain of telemedicine requires strict, very specific guidelines to protect sensitive data. The reason why ABAC and RBAC by themselves are not enough is that the requirements for interoperability and interchangeability demand a much more dynamic and fine-grained approach. The PEP should support the formulation and assessment of even the most specific needs.

Our work seeks to combine these two approaches while separating the access control process from the backend and frontend and putting it on the path of the data between the cloud and the end users. In our previous study [33] we introduced the concept of a hybrid access control methodology, taking the classical roles of the telemedicine environment and assessing its requirements based on the content of the documents and the contextual information. Later, we described a theoretical infrastructure [34] which, based on our assumptions, should allow us to effectively implement this methodology in a clinical environment. In the latter case, we not only defined the four main categories of required policies, but also selected a potential candidate as our policy enforcement point and performed various tests on FHIR documents while monitoring CPU load and memory usage. These early evaluations demonstrated that our concept is viable.

Since then, we have been able to develop a prototype of this infrastructure, which integrates the chosen PEP between a small client application and a scalable NoSQL database with over 500,000 FHIR documents.

In this paper we investigate the performance and relative latency of a PEP engine as part of the larger infrastructure. To this end, in the Related Work section we give an overview of several possible solutions and research projects that, with at

least some degree of similarity to our study, also sought to combine access control methods and define complex, interoperable security frameworks for health care. In the next section we describe our results obtained so far in greater detail and present the test environment and the various policies for the evaluation process. Then, in the Results section we share and analyze the results of our tests to determine trends in the evaluation process and possible patterns, anti-patterns for our future work. Finally, in the Conclusions section we summarize these results and provide an outlook on future steps and possible new directions for research.

2 Related Work

While the authors of a 2013 comparative study based on 775 reviewed articles found that RBAC [13] was the most popular approach to manage access control in healthcare, this trend changed significantly with FHIR, and the preference between ABAC and RBAC became the de facto choice of the development team rather than industry standards.

For example, the developers of the application atHealth [31] succeeded in implementing a role-based methodology for their mobile application in 2017, recognising the lack of security in FHIR. However, there also were implementations of the ABAC model for access control to health records [21] in the same year.

To further complicate the issue of these two models, as early as 2008 [23] there were critics who noted that access control in healthcare systems is sufficiently complex to justify situation-based decisions, with the classical concept of roles and attributes oversimplifying the issue. When we conducted our first experiments in 2018 in this area [33], we also found that neither ABAC nor RBAC as such is sufficient to meet the needs of practitioners and clinical applications, because even though roles are important elements of security, they cannot cover every situation without specific, contextual information.

Although there are platforms, such as the popular SMART project [11], which offer a solution in the form of a full OAuth2 integration into their FHIR database, the use of such frameworks usually comes at the expense of a certain degree of freedom in the choice of health infrastructure components. There have been several attempts to define hybrid solutions both in healthcare [24] and more generally in multi-modal, heterogeneous environments [9]. A key concept of the domain is the requirement to control access not only to entire documents, but also to specific fields and attributes in documents. The proposed architecture by Rezaeibagha, F. et al [28] is specifically designed to move sensitive data from a secure private cloud to a public one, while maintaining security.

In 2016, Pusselwalage H. S. G. et al. [25] published their approach for an ABAC methodology that bases its policies not only on the attributes of the data but also on the attributes of the user, treating the different levels of access and the classical roles in healthcare as attributes. They combined the two models to some extent, while also highlighting special cases such as unregistered users or registered users without a specific role. In 2018 Joshi M. et al. [20] used a similar approach with roles treated

as attributes, but instead of granting full access, their solution also transformed the requested data to match the requester's access level. The developers of the SOCIAL platform [29] also discussed some interesting ideas about treating the requesting device as an important component of ABAC with a combination of the user attributes.

During our review we also found some studies that appeared to combine elements of the RBAC and ABAC models without clearly categorizing their methodology as a hybrid. The developers of the [26] H-Plane Framework, which follows the terminology of the ABAC model, also apply several attributes in a way that is almost identical to some aspects of RBAC. In their publication they also pointed out the importance of the IoT in this domain. In 2019, Alnefaie, S. et al. [6] after reviewing the possible alternatives for access control they thought ABAC was much better suited to the needs of healthcare in combination with edge computing, but also suggested modifying the infrastructure of the methodology to bring the point of evaluation closer to the edge and place more emphasis on the identity of the IoT device itself. Tasali Q. et al. [35] extended this concept by covering not only medical data, but also the authorization process for real-time communication between IoT devices.

The solution proposed by the developers of the mHealth application [22] is also quite similar to ours, the main differences being that its policy engine is deployed as part of the cloud services and the engine is an implementation of the NIST NGAC framework, with the evaluation process based on traversing a Neo4j graph database. The infrastructure and principle designed by Ray, I et al. [27] also have similar features, with policy enforcement based on the XACML format.

To summarize the state-of-the-art based on these sources:

- A modern solution should either extend the traditional access control ABAC model or develop a custom hybrid solution to meet the needs of the domain;
- Heterogeneous storage should be taken into account and the sensitive documents must be transformed before they enter the public cloud;
- The IoT raises brand new challenges. The security solution must be able to handle the different capabilities and requirements of these tools when evaluating and converting the healthcare data.

It is clear that our approach is only one of many proposals that seek to resolve the security issue of EHR. Our goal is to combine the best ideas and elements of the domain - combining RBAC and ABAC, establishing the included PEP nodes as a middle layers between the private and public clouds, public cloud and edge network, etc. - and also to improve and extend them, to provide support for every database and application that uses FHIR, and to provide users and developers with a trusted, verified solution to the security problem.

3 Our Approach

3.1 Telemedicine Security Infrastructure

The goal of our research is to create a solution that is able to support several different storage providers and at the same time make the infrastructure shown in Figure 3 transparent to the end users. While the use of such heterogeneous backends is recommended in various use cases, the field of telemedicine is the prime example of how the strengths of this model can be brought to bear. Even before the GDPR, the storage and encryption of data was a major challenge, and while public cloud providers proved to be very popular with telemedicine developers, the storage in such solutions required high-level encryption and data transformation. However, with heterogeneous storage, it is possible to store the sensitive data in a private, more secure cloud and the less sensitive information in a public cloud or even in a custom database to improve and optimize the efficiency of the system as a whole. To achieve this, we needed to place our entire policy enforcement flow - the Policy Enforcement Point, Policy Decision Point, and Policy Identity Point - between the back-end and front-end and connect it to a proxy. We chose a Squid proxy [30] implementation to achieve the latter and configured it to forward each request, but upon receipt of a response containing FHIR structures, send it to the PEP for filtering and (if necessary) transforming before forwarding it to the end user. This provides a necessary middlelayer, unlike most solutions we discussed in the previous section. In this way, policy enforcement can take place outside the cloud, which allows the use of a heterogeneous storage solution (provided that it uses the FHIR standard as the format of the stored documents), but it also relieves the burden on the end systems. This approach is not only better optimized in terms of efficiency and capacity, as some of the end systems may not have the required capabilities, but it also ensures that sensitive information never reaches the end users without prior assessment and filtering. It should be added that with such a proxy, developers are also able to log in detail the various operations on the telemedicine records in order to comply with the GDPR.

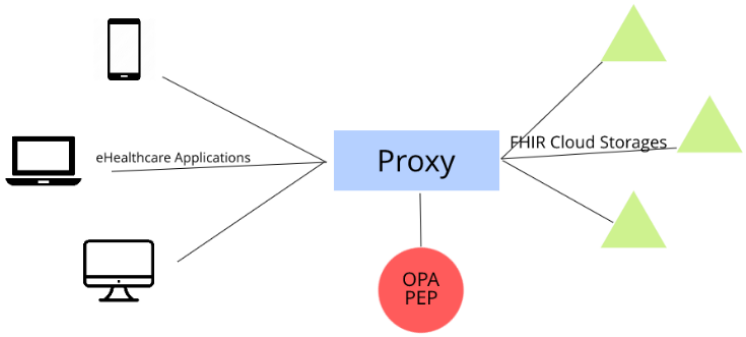

Figure 3: A general outline of our proposed solution

To test our concept of policy enforcement, we chose a promising new solution called Open Policy Agent [4], available in Go and WebAssembly, which could play every role in the enforcement process. There is also an OPA implementation in WebAssembly - this way, if it turns out to be an effective PDP/PEP solution, it could be placed on any part of the infrastructure (or even in multiple locations simultaneously) to meet another one of our requirements. OPA also permits us to store the information necessary for decision making in JSON format and define the various policies in its own scripting language, Rego, which can later be accessed via a well-defined REST interface with an HTTP POST request containing the contextual information to be filtered or evaluated (in our case, the medical records).

3.2 Test Environment

At this stage of our research, we decided to use the cleanest possible test environment. To achieve this, we created the prototype infrastructure on a local network connected via WiFi (at 5 GHz frequency with only the elements of the prototype infrastructure connecting) to the various nodes hosting different actors of our model, instead of testing with a well-known cloud provider like Google Firebase. This means having:

- A desktop PC running Windows 10 on an AMD Ryzen 5 processor at 3.59 GHz speed and 16 GB DDR4 memory ran the client application on a Kingston SSDNow V300 SSD with 120 GB capacity and 450 MB/s reading speed, acting as the controller node of the environment;
- A laptop running Windows 10 on an Intel i5 processor at 2.49 GHz speed with 8 GB DDR4 and a 120 GB SSD with a reading speed of 423 MB/s memory hosted the MongoDB v3.2.1 [2]-based backend along with a lightweight REST API that handled the requests and query parameters;
- A secondary laptop with similar attributes hosted the Squid proxy written in NodeJS 10.14;
- An iMac running macOS Catalina with an Intel i5 processor at 2.9 GHz speed and 20 GB DDR3 memory hosted the OPA v0.23.2 runtime with a hard disk of size 1 TB and reading speed of 210 MB/s.

The database was loaded with over 500,000 different FHIR Observation documents, based on properties from the MIMIC3 database [19], involving 200 patients, 30 doctors and 12 nursing teams, all signed with a different time stamp between 2015 and 2020. The template and structure of these Observations were taken from one of our industry projects to simulate the size and complexity of healthcare documents in a real system. The measurements were performed by a monitoring host that issued the restarts and reinitializations of each element of the architecture between measurements. During the experiment, each component was configured so that its output was logged in separate files that were collected and evaluated by

the monitor at the end of a round. Each rule ran on 8 different input sizes: 10, 20, 50, 100, 200, 500, 1000 and 2000 data sets. The PEP engine received exactly the same amount of data in each round, but of course the size of the output varied from case to case, depending on the selected policy and the content of the input.

3.3 Evaluation Role Set

In our previous paper [34], we defined the four main categories of policies required by the domain to evaluate health data before it reaches the external network and end-user applications. We based this categorization on several sources and overviews of the domain [8] [17] [16] [5], from which we were able to determine the basic and extended safety requirements of the health sector. In accordance with the GDPR, every user of the system must naturally have full control over their data. The patient is the primary owner, the physician who wrote the document or assisted in its creation is the secondary, while other practitioners and relatives can have access to it to some extent. The system must also handle indirect access when the applicant, as a member of a group, tries to access the file. These respective types of access must be identified based on a combination of user roles, role groups and the attributes of the FHIR documents.

In some cases, contextual information is also required to determine the degree of access. For example, while a general practitioner should be able to access patient records at any time (logging the exact time and nature of such access), a nurse or assistant should not allowed to exceed the prescribed office hours. For some especially sensitive information, other contextual information such as the physical location of the requester, the ID of the device from which the request originates, should also be used in the evaluation, and expanding this set compared to a simple role definition is enough to justify a separate category.

A key requirement in the field of healthcare is that access does not mean full access to every element of the given document. In many cases it is strictly forbidden to grant access to such information from which a third party might be able to reconstruct very sensitive events and elements. For example, if one receives a list of a patient's medicines from a certain period of time, it is easy to infer vital information that would otherwise be prohibited for that particular third party. The evaluation process in a healthcare environment should be able to determine access at a very fine granularity, essentially at the field-by-field level, and to mark or even remove certain fields that should not be available at that security level. This is also the reason why the standard security solutions of several large cloud providers and databases fails, as they can only provide this functionality by including lambda functions, trigger functions, and the like.

The last requirement is also the most unique and difficult aspect of healthcare security. The break-the-glass case requires an access control model that provides immediate access to key patient information to ensure the receiving of the necessary, possibly life-saving care. This is essentially what happens in an emergency, when life-saving surgery is required and neither the patient nor the doctor recording and processing their health data is available to grant access. In a break-the-glass

situation usually only a few records are required, but in that case it is important to use very complex transformations. Only vital information should be accessed, while every other element of the document must be either removed or encrypted. Without the effective implementation of break-the-glass, no healthcare security system can be used in real-life situations.

Based on these requirements, the definitions of our policy categories are:

- **Role Evaluation**: The policy has to determine whether based on the user's role or roles in the system, partial or full access should be provided;
- **Contextual Evaluation**: The policy has to determine whether the combination of the user's role, various attributes and contextual information form the basis for partial or full access;
- **Contextual Modification**: Aside from providing access, the policy should also transform the data, removing or altering specific fields;
- **Break-the-Glass**: A specific requirement of a healthcare application. In the case of an emergency, the policy should provide immediate access, while also encrypting or removing sensitive information.

Our first step during the evaluation process was to test our evaluation concept with different policies and different pressures. During this first phase we ran our tests on OPA without including other elements of the proposed infrastructure and stored the FHIR documents in its database, making it essentially a temporary FHIR database. We measured attributes such as CPU load and memory usage, while increasing the size of the input data set by a power of ten after each iteration, up to a data set of one million records. The results of these experiments demonstrated that the combined PDP/PDE/PIP concept was an acceptable candidate.

These results paved the way for the next step of our research: the integration of the combined PDP/PDE/PIP node (OPA) with a prototype infrastructure and the further evaluation of its effectiveness and latency in this environment.

3.4 The Telemedicine Security Abstract Role Set

First and foremost we defined a formal specification of each category, with the following notations:

- $\mathbb{F} := \{f_1, ..., f_k\}$ marks the telemedicine record in question, where each f_x is a valid key-value pair of the record.
  For example:
  $\mathbb{F} := \{('subject', 'PAT/1'), ('systolic\_bloodpress', 120), ...\}$
- $\mathbb{UR} := \{r_1, ..., r_l\}$ where $\mathbb{UR} \subseteq \mathbb{F}$ is a subset containing the key-value pairs describing various primary or secondary owners of the record
  For example:
  $\mathbb{UR} := \{('subject', 'PAT/1'), ('practitioner', 'PR/A013'), ...\}$

- $\mathbb{EX} := \{e_1, ..., e_m\}$ marks the external context of the system at the time of the policy evaluation as key-value pairs
  For example:
  $\mathbb{EX} := \{('datetime',' 2020{-}09{-}12T12:20:33'), ('ip\_addr',' 223.134.22.1'), ...\}$
- $\mathbb{CX} := \{c_1, ..., c_k\}$ is a set of conditional functions, which take an atomic value as an argument and transform it to a boolean value. Each function is represented as an (op, val) pair where op is a conditional operation, $op \in \{<, >, \leq, \geq, =\}$ and $c_i(n) := n\ op_i\ val_i$
  For example:
  $c_1 := (>, 12), x := 5 \implies c_1(x) := 5 > 12 \implies c_1(x) :=$ false
  $c_2 := (=,' bloodpressure'), x :=' bodyweight' \implies$
  $c_2(x) :=$ 'bodyweight' = 'bloodpressure' $\implies c_2(x) :=$ false
- $\mathbf{P}(n)$ is a function describing a policy to be enforced by a PEP engine, where $f_x \in \mathbb{F}$ and $\mathbf{P}(f_x) = Allow|Modify|Deny$ produces the decision regarding the evaluated key and $\mathbf{P}(\mathbb{F}) := \{\mathbf{P}(f_1), ..., \mathbf{P}(f_k)\}$

3.4.1 Formal Definition of the Role Evaluation Policy

Definition 1. *$\boldsymbol{P}(n)$ describes a Role Evaluation policy, if $\mathbb{UR} \neq \emptyset$ and for a given user identifier $\exists key(key, id) \in \mathbb{UR}$, then $\boldsymbol{P}(n) := \forall f_i \in \mathbb{F}\ \boldsymbol{P}(f_i) := Allow$, else $\boldsymbol{P}(n) := \forall f_i \in \mathbb{F}\ \boldsymbol{P}(f_i) := Deny$*

3.4.2 Formal Definition of the Contextual Evaluation Policy

Definition 2. *$\boldsymbol{P}(n)$ describes a Contextual Evaluation policy if $\mathbb{G} := \mathbb{F} \cup \mathbb{EX}$, $\mathbb{CE} := \{ce_1, ..., ce_x\}$ is a set of contextual conditions where $0 \leq i \leq x$, $ce_i := (key_i, c_i)$, $c_i \in \mathbb{CX}$ and $\exists x : (key_i, value_i) \in \mathbb{G}$ and $\boldsymbol{P}(n) := \forall f_i \in \mathbb{F}\ \boldsymbol{P}(f_i) := Allow$, if $\forall x : (key_i, c_i) \in \mathbb{CE} : \exists (key_i, value_i) \in \mathbb{G}$ and $c_i(value_i) := true$, else $\forall f_i \in \mathbb{F}\ \boldsymbol{P}(f_i) := Deny$*

3.4.3 Formal Definition of the Contextual Modification Policy

Definition 3. *$\boldsymbol{P}(n)$ describes a Contextual Modification policy if similarly to the Contextual Evaluation policy, $\mathbb{G} := \mathbb{F} \cup \mathbb{EX}$, $\mathbb{CE} := \{ce_1, ..., ce_x\}$ is a set of contextual conditions where $0 \leq i \leq x$, $ce_i := (key_i, c_i)$, $c_i \in \mathbb{CX}$ and $\exists x : (key_i, value_i) \in \mathbb{G}$ but there is also a $\boldsymbol{FX}$ mapping, which $\boldsymbol{FX} : \mathbb{CE} \implies \mathbb{F}' \subseteq \mathbb{F}$, with $\cap_{0 \leq i \leq x}\ \boldsymbol{FX}(ce_i) := \emptyset$. $If 0 \leq i \leq x\ ce_x(g_x) := false$ then $\forall f_i' \in \boldsymbol{FX}(ce_i)\ \boldsymbol{P}(f_i') := Deny$, else $\forall f_i' \in \boldsymbol{FX}(ce_i)\ \boldsymbol{P}(f_i') := Allow$*

3.4.4 Formal Definition of the Break-the-Glass Policy

Definition 4. *$\boldsymbol{P}(n)$ describes a Break-the-Glass policy if $\boldsymbol{P}(n)$ satisfies the requirements of the Contextual Modification with the further addition of a $\boldsymbol{TX}$ mapping, identifying the attributes which have to be encrypted or modified $\boldsymbol{TX} : \mathbb{CE} \implies \mathbb{F}'' \subseteq \mathbb{F}$, with $\cap_{0 \leq i \leq x}\ \boldsymbol{TX}(ce_i) := \emptyset$ and $\cup_{0 \leq i \leq x}\ \boldsymbol{FX}(ce_i) \cap \cup_{0 \leq i \leq x} \boldsymbol{TX}(ce_i) := \emptyset$.*

$If 0 \leq i \leq x\ ce_x(g_x) := false\ then\ \forall f_i'' \in \boldsymbol{FX}(ce_i)\ \boldsymbol{P}(f_i'') := Deny,\ else\ \forall f_i'' \in \boldsymbol{FX}(ce_i)\ \boldsymbol{P}(f_i'') := Modify$

3.5 Implementation Details

To achieve this goal, we defined a new set of guidelines based on the actual needs of a healthcare system, instead of the proof-of-concept drafts from our previous study. We defined two rules for each of the four categories - one simpler and one more complex, the latter containing more operations or more resource-intensive operations, or both. All algorithms run on multiple Observations received as part of the input, but only returned those in their original or modified state which were allowed by the policy.

The two policies of the Role Evaluation category included in algorithms 1 and 2 both focus on the identity of the practitioner. However, while *role_simple* only checks to see that the specified role identifier matches the practitioner's identifier in the document, *role_complex* checks the care teams responsible for the patient and only grants access if the requester is a member of those teams.

The main difference between the policies of the Contextual Evaluation category, shown in algorithms 3 and 4, is the nature of the contextual attribute.

In *context_simple* we check a high-level attribute, the status of the Observation and an external attribute, the hour. Here *context_complex* requires the PEP iterating through the component array of each Observation, finding each medical value based on the defined LOINC identification code, and checking to see if the exact value is greater than the threshold.

A key aspect to be evaluated was the efficiency of the PEP when iterating and handling arrays, since in the current version of the OPA, the developers noted in the official documentation [3] that the performance of such an evaluation engine is the most efficient when it works with objects, and weakest when it must iterate through non-indexed arrays.

Algorithm 1 A Role Evaluation policy in Rego returning only the Observations where the current user is the Practitioner

Policy role_simple[*shell*]

1: $pract := input.practitioner$ ▶ We get the Practitioner id from the request
2: $shell := input.observations[\_]$ ▶ We create a working copy from the list of Observations
3: $observation := shell.data$ ▶ We iterate through the shell array and map the Observation inner object
4: $performer := observation.performer[\_]$ ▶ We map the list of Practitioners of the current observation
5: $performer.identifier.value == pract$ ▶ We check if one of the Practitioners has the same id as the input. If so, it remains in the shell array and will be returned, if not, it will be filtered out

Algorithm 2 A Role Evaluation policy in Rego returning only the Observations where the current user is member of one of the CareTeams, who received access from the Patient

Policy role_complex[*shell*]

1: pr := *input.practitioner* ▶ We get the Practitioner id from the request
2: *shell* := *input.observations*[_] ▶ We create a working copy from the list of Observations
3: *observation* := *shell.data* ▶ We iterate through the shell array and map the Observation inner object
4: $performer$:= *observation.performer*[_] ▶ We map the list of Practitioners of the current observation
5: *cteam* == *performer.identifier.value* ▶ We get the identifier of the Performer
6: contains('CareTeams', *cteam*) ▶ We call the built-in function to check whether the identifier belongs to a CareTeam, and only to proceed if it is true. If the Observation has no CareTeam at all, the evaluation returns false.
7: count(practition_member_of_careteam(pr, *cteam*)) > 0 ▶ We call a simple function which checks whether the identifier of the Practitioner is listed as a member of a the CareTeam. If there is a match for even one of the CareTeams, to access is provided

Algorithm 3 A Contextual Evaluation policy where access is granted only if the status of the Observation is active and the time of access takes places between 8:00 and 19:00

Policy context_simple[*shell*]

1: *timearray* := time.clock([time.now_ns(), 'Europe/Budapest']) ▶ Using the built in functions of Rego, we generate the array containing the parts of a time string
2: *hour* := *timearray*[0] ▶ We retrieve the first element of the timearray which will always be the hour from the chosen timezone
3: *shell* := *input.observations*[_]
4: *shell.data.status* == 'active'
5: *hour* $\geq$ 8
6: *hour* $\leq$ 19 ▶ These last three lines are executed at the same time, if all of them return as true, the Observation will be part of the result set

Algorithm 4 A Contextual Evaluation policy where the internal structure of the Observation is analyzed and only a certain type with its value above a predetermined limit can be accessed

Policy context_complex[*shell*]

1: *shell* := *input.observations*[_]
2: *component* := *shell.data.component*[_] ▶ We iterate through the inner attributes of the Observation
3: *component.code.coding*[0].*code* == '32419-4'
4: *component.value* > 5

Algorithm 5 A Contextual Modification policy where we remove the Patient identifiers from Observation

Policy modif_simple[*shell*]

1: *observation* := *input.observations*[_]
2: *clean* := object.remove(*observation*, ['patient', 'data']) ▶ We remove the external patient identifier and the entire nested data object - since in the current version of Rego we cannot modify existing key-value pairs only remove existing or add new ones
3: *inner_new* := object.remove(observation.data, ['subject']) ▶ We create a new nested data object without the subject fields
4: *shell* := object.union(*clean*, {'data': *inner_new*}) ▶ We create the result Observation by merging the two filtered versions

Algorithm 6 A Contextual Modification policy where we remove a specific nested component from every Observation

Policy modif_complex[*shell*]

1: *observation* := *input.observations*[_]
2: *clean* := object.remove(*observation*, ['data'])
3: *no_components* := object.remove(*observation.data*, ['component']) ▶ We need to filter the nested object as well
4: *new_components* := [*component*|
5: *component* := *observation.data.component*[_];
6: *component.code.coding*[0] != '18748-4'] ▶ We filter the nested object based on the LOINC codes
7: *new_data* := object.union(*clean*, {'component': *new_components*})
8: *shell* := object.union('*new_data*', {'data': *new_data*})

The majority of the documents in the FHIR standard use the array structure very often, and if the performance of array operations is significantly worse than in any other case, this would provide a strong argument against adapting our concept. The modification operation in a PEP node is very demanding in itself, since the engine treats every variable as a constant due to their non-imperative behavior. For this reason, if we need to modify or redefine a particular field, we must first create a copy of the original object without the value, then create a new value, and finally create the response by adding the new value to the filtered object copy. The Contextual Modification policies are shown in algorithms 5 and 6. Here *modif_simple* simply removes the patient data from the Observation and the encapsulating shell, while *modif_complex* must create a new component array for each Observation that does not contain urls pointing to sensitive patient documents.

The Break-the-Glass Policies shown in algorithms 7 and 8, are the most complex. These policies are expected to be fast, accurate, and effective, because they are most commonly used in emergency scenarios when a doctor or nurse needs to access limited patient information to provide the necessary care.

Algorithm 7 A Break-the-Glass policy where we hash the identifier of the CareTeam in the Observations

Policy break_simple[*shell*]

```
observation := input.observations[_]
clean := object.remove(observation, ['data', 'careteam'])
no_performer := object.remove(observation.data, ['performer'])
doctors := [performer| ▶ We create a performer array without the CareTeams
    perfomer := observation.data.performer[_]
    performer.type == 'http://hl7.org/fhir/practitioner.html']
hash_careteams := [perf| ▶ We create an array from the hashed CareTeams
    performer := observation.data.performer[_]
    not performer['type'] ▶ CareTeams do not have the optional type attribute
    perf := {'identifier': {'value': crypto.md5(performer.identifier.value)}}]
new_performers := array.concat(doctors, hash_careteams)
new_data := object.union(clean, {'data': new_data, 'careteam': {'identifier':
    {'value': crypto.md5(observation.careteam.identifier.value)}})
```

Algorithm 8 A Break-the-Glass policy where we hash every identifier for Patients, Practitioners and CareTeams

Policy break_complex[*shell*]

```
observation := input.observations[_]
clean := object.remove(observation, ['data', 'careteam', 'practitioner', 'pa-
    tient'])
removed_body := object.remove(observation.data, ['performer', 'subject'])
new_body := object.union(removed_body,
    {'subject': {'display': crypto.md5(observation.data.subject.display), identi-
    fier: {'value':
    crypto.md5(observation.data.subject.identifier) }},
    'performer': [perf|
      performer := observation.data.performer[_]
      perf := {'identifier': {'value': crypto.md5(performer.identifier.value)}}]})
      ▶ We also tried whether a nested instruction set would increase or stabilize
      resource consumption
shell := object.union(clean,
    {'data': new_body, 'patient': crypto.md5(observation.patient),
    'practitioner': crypto.md5(observation.practitioner),
    'careteam: {'identifier': {'value':
    crypto.md5(observation.careteam.identifier.value)}}})
```

In these scenarios, the system must remove or encrypt everything else that goes beyond the most necessary attributes, and with two policies we tested how resource consumption varies when we wish to encrypt a single attribute that is deeply embedded in the document and requires filtering in *break_simple* and in

break_complex when we wish to encrypt every identifier in the document that could later be used to identify users in the system.

4 Results

4.1 PEP Performance between Categories

After evaluating the average performance of the various categories, we observed several interesting trends, compared to what we originally expected. The most notable of these is the relatively faster evaluation time of the Contextual Evaluation policies compared to the Role Evaluation category, as seen in Figure 4.

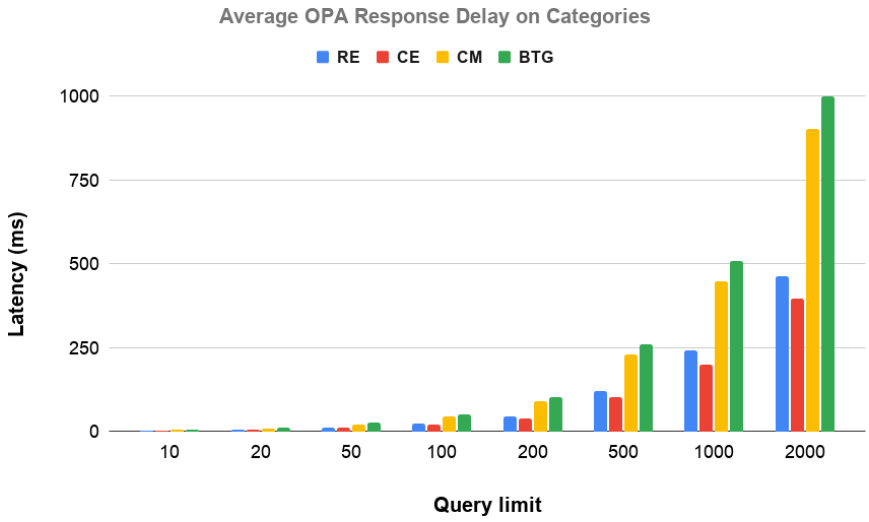

Figure 4: Average Delay on PEP by Categories

While the CE policies are more complex in nature, it seems that if the contextual information sets the result as true or false, the evaluation is significantly quicker than the cases when an internal examination of the input documents is required.

We observed a similar trend with the average CPU load of the categories, shown in Figure 5 with the Contextual Evaluation policies demanding slightly less percentage of the CPU time compared to the Role Evaluation policies, while the Break-the-Glass policies remain the most demanding ones. However, the overall difference between the first two and latter two categories is not as big as on the response delay. Another key observation is that while the size of the input was below 1000 documents (which is already an unrealistically large query size for a real-life application), not even the Break-the-Glass policies required more than 50% of the CPU.

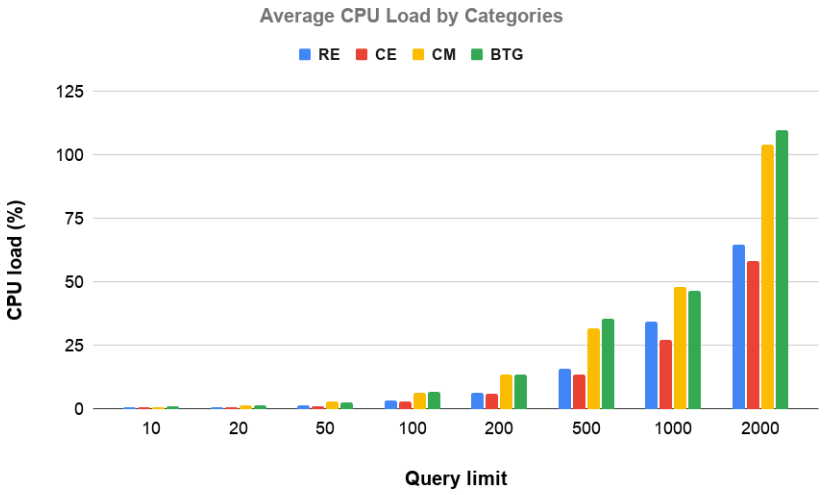

Figure 5: Average CPU Load of OPA by Categories

The memory usage of the categories, shown in Figure 6 on the other hand, while still showing the trends of the previous figures and requiring only manageable amount of memory when evaluation smaller inputs (not even Break-the-Glass policies demanding more than 50-60 MB while the input size is around 50 documents), this demand shows a sudden jump after the input size reaches 1000 documents, with even the Role Evaluation policies requiring around 100-150 MB to evaluate inputs between sizes 1000 and 2000.

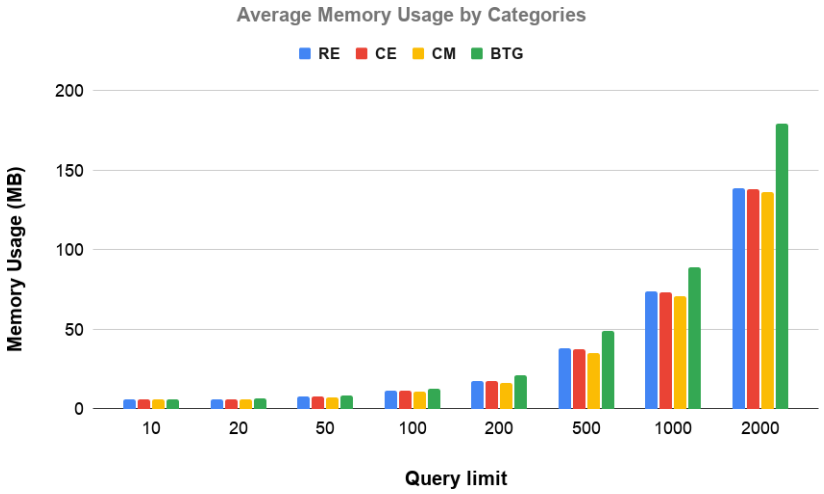

Figure 6: Average Memory Usage of OPA by Categories

The measurements were taken with the PEP solution restarted and reinitialized between each measurement, since, as we have shown in our previous paper, OPA employs a very lazy approach towards garbage collecting, cleaning the memory only when it is required by the system or an especially large evaluation. This fact makes these sizes even more alarming for a real-life scenario, since the size of OPA in the memory can grow significantly during a series of evaluations.

Based on these results, while the CPU load and the response delay seem to be manageable requirements, the memory demand, combined with the experienced lazy garbage collecting process of OPA might requires a custom build or external process that manages and frees the memory after the evaluations are finished to optimize this aspect of the PEP nodes.

4.2 PEP Performance in Categories

We ran each policy with each size at least 30-50 different times to collect the raw data for the statistics shown in the tables below, and these group the policies belonging to the same category. For each policy, we calculated from the collected data sets the mean value of CPU load, memory usage and response delay on the PEP (OPA) node.

Although the question of whether to use the same constant inputs for each evaluation, or use HTTP(S) requests that simulate a real-world application is a complex element for this phase of our research, we decided to use dynamic inputs to get more precise, realistic results.

4.2.1 Role Evaluation Policies

A comparison between Role Evaluation policies is shown in Figure 7. While, as we have assumed, the complexity of *role_complex* induces a higher latency, the total difference between the two policies is not very significant. Even with an input of 2000 records the PEP was able to filter out the restricted ones in half a second.

The effects on CPU and memory, as shown in Figures 8 and 9, are somewhat more demanding – when 2000 documents are sent, 70% of the processor is required to evaluate the policy and about 140 MB of the memory – a clear indication of how costly it is to perform subqueries in isolated structures, such as the list of careteams and their members.

Based on these results, it is evident that the proposed solution is capable of handling more complex Role Evaluation policies without difficulty, but it is advisable to store the teams, groups, institutions in indexed objects rather than in arrays.

4.2.2 Contextual Evaluation Policies

A comparison of the two Context Evaluation policies also produced some interesting results, which are presented in Figures 10, 11 and 12. Our main objective here was to determine what kind of contextual evaluation is more demanding, and on the basis of the data it is clear that array-based evaluations are generally more complex,

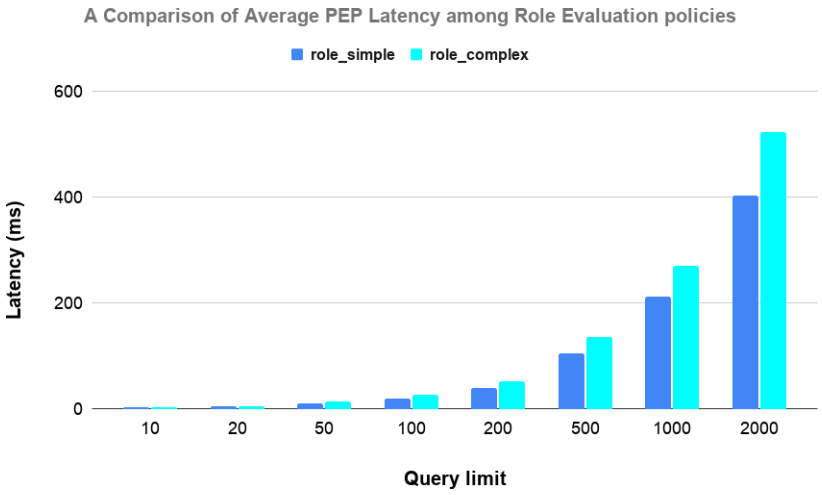

Figure 7: Average PEP Latency in Role Evaluation category

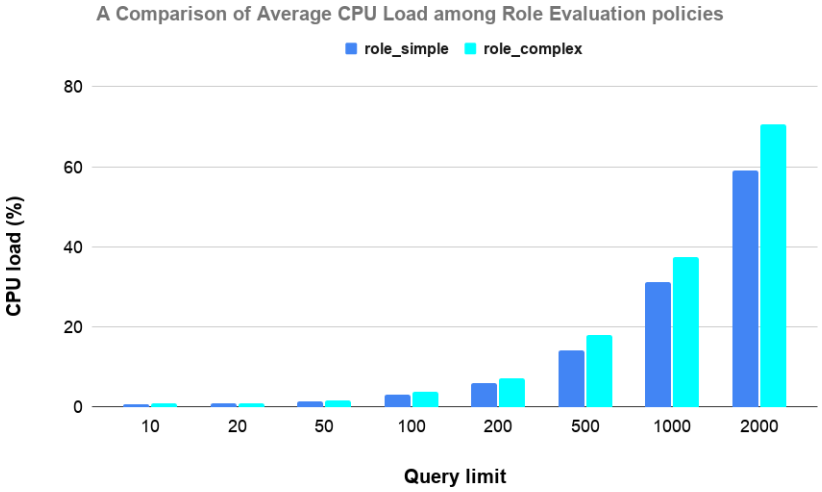

Figure 8: Average CPU load in Role Evaluation category

but in small evaluations they are actually cheaper than collecting and comparing external information such as dates.

This calls into question some notable architectural aspects of the infrastructure, such as whether this context information should be collected and forwarded by the proxy as part of the input data. Seeing that in some cases it is possible on a larger scale that the PEP deployment can handle traffic from end nodes in different time zones, it might be a good idea to omit such internal queries as a general design pattern of the policies.

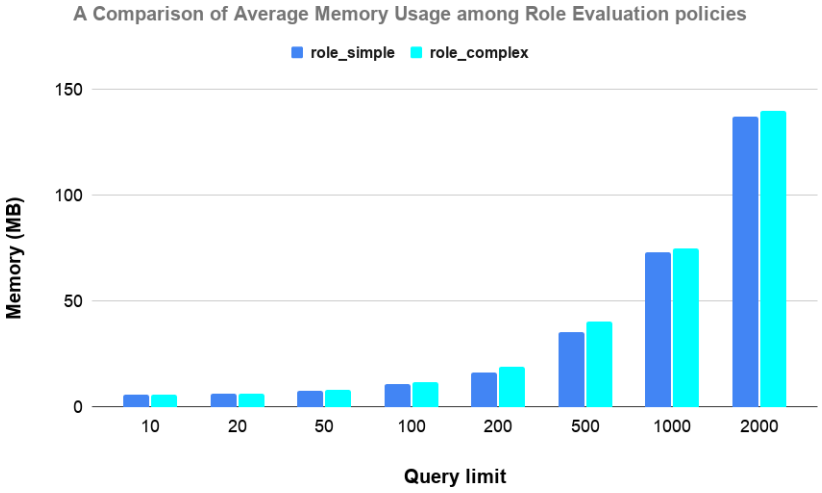

Figure 9: Average Memory usage in Role Evaluation category

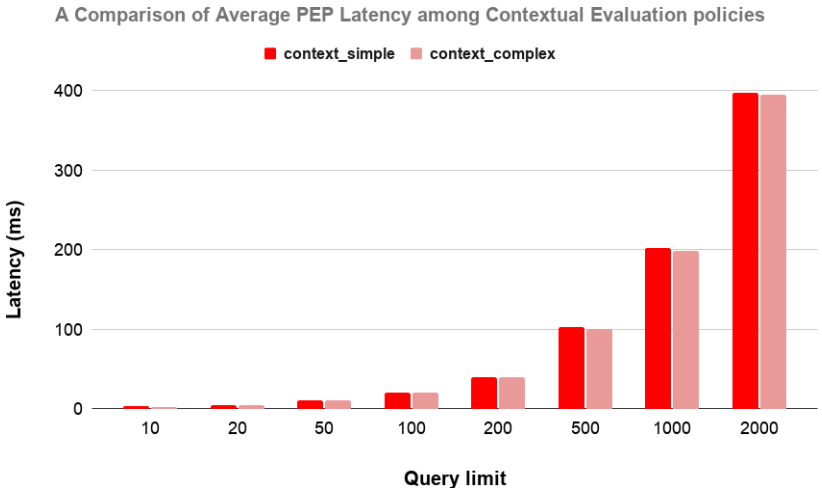

Figure 10: Average PEP Latency in Contextual Evaluation category

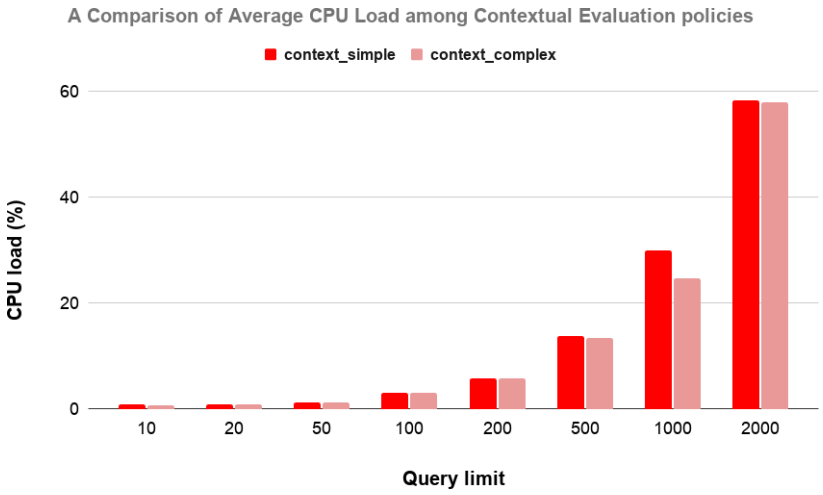

Figure 11: Average CPU load in Contextual Evaluation category

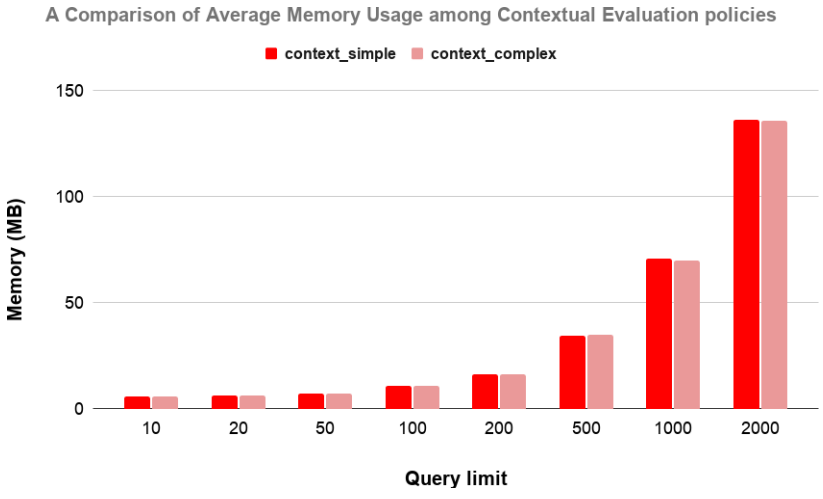

Figure 12: Average Memory usage in Contextual Evaluation category

Moreover, it is interesting to note that after some minor differences, besides inputs with more than 20 documents, the metric values start to converge, since after a certain point in the measurement set a significant portion of the documents is rejected during the evaluation of context1 due to their inactive status, thus the value is set false, before the engine starts evaluating the relational conditionals.

Our interpretation of these results can be summarized as follows: It is clear that policies with contextual evaluation may be as effective as simple Role Evaluation policies, but where possible, contextual information must be provided as part of the input, rather than being queried on the PEP node.

4.2.3 Contextual Modification Policies

The results of the Contextual Modification policies are included in Figures 13, 14 and 15. These results showcased another important aspect of the evaluation engine that could be one of the pillars of the design patterns for defining the policies. It was expected that *modif_complex* would be the more complex of the two.

Instead of this expected result, the measurements clearly indicate that neither is significantly more demanding. In some cases *modif_complex* consumes more CPU, and it has slightly more latency than *modif_simple* due to the demanding array copy and filter mechanisms, while *modif_simple* requires slightly more memory.

Modification policies are much more demanding than simple access evaluations. Nevertheless, they can be implemented effectively if we take into account the increased costs. We also wish to investigate the possible patterns and anti-patterns in order to write more effective policies for this type.

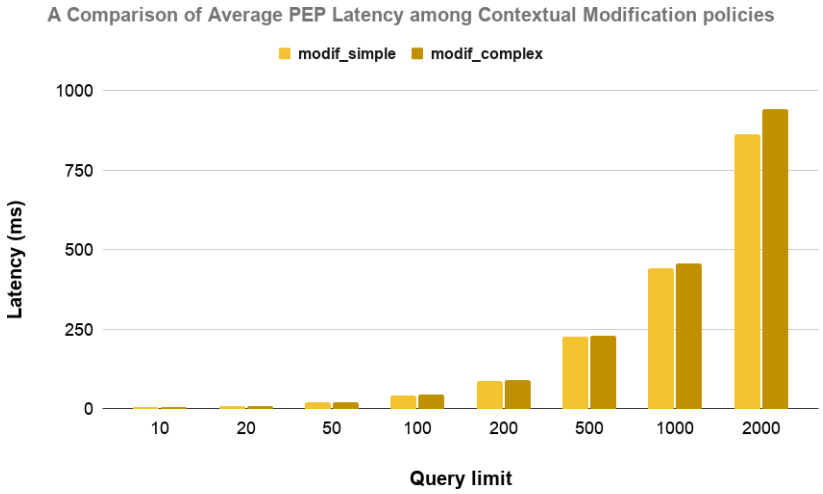

Figure 13: Average PEP Latency in Contextual Modification category

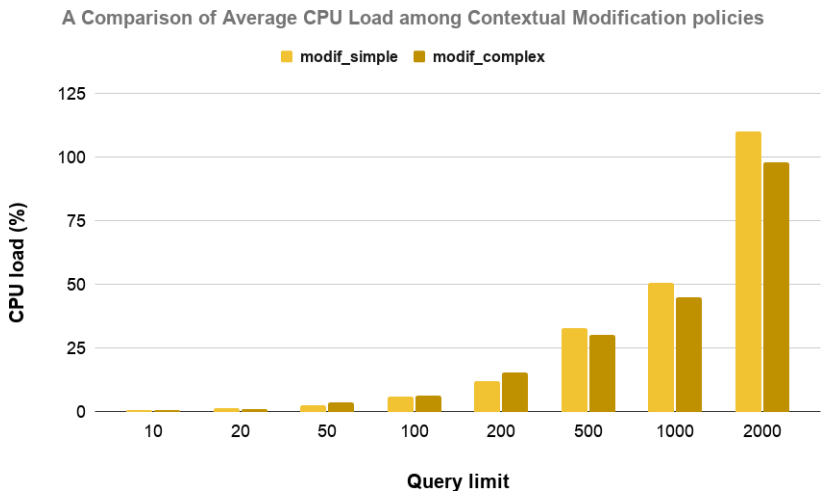

Figure 14: Average CPU load in Contextual Modification category

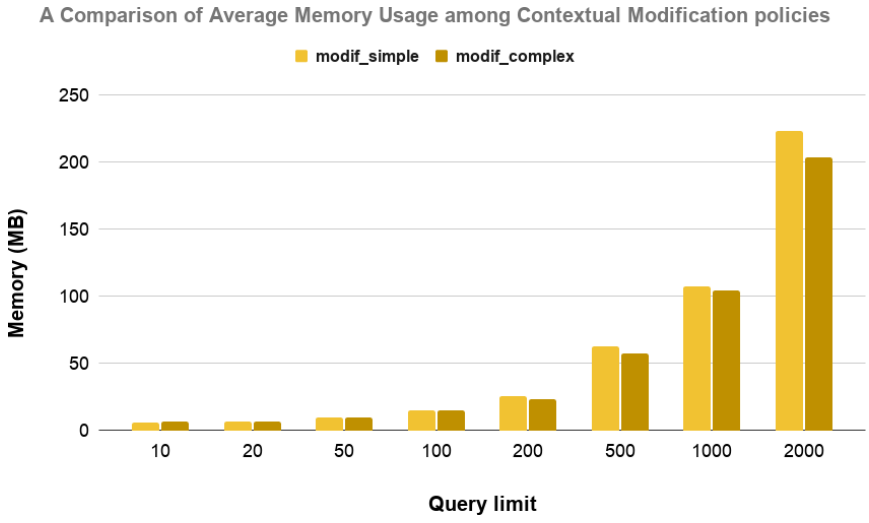

Figure 15: Average Memory usage in Contextual Modification category

4.2.4 Break-the-Glass Policies

Based on the results of our previous evaluations, we assume that Break-the-Glass policies will be the most resource-intensive portion of our evaluation set, and the results (see Figures 16, 17 and 18) were as we thought they would be, but again with some minor differences from our original expectations.

While the CPU load and memory usage of the two policies are almost identical, it actually takes a little longer to evaluate *break_simple* than *break_complex*, although based on the sheer number of operations (especially the number of encryption operations) in *break_complex*, it should have been a much more expensive policy compared to *break_simple*. The answer lies in the nature of operations that the policy executes: It filters an array, then creates a new array to store an encrypted value, and then concatenates two arrays to be embedded in an object. Apparently, these array operations, with the emphasis on the concatenation operation, which is unique in our evaluation set for this policy, is just as resource-demanding as its pair.

Just as we expected, the Break-the-Glass policies are the most demanding ones that can be evaluated on the PEP node, but even with the increased cost they can achieve the expected results. While the essence of these policies is to transform and encrypt the data, it is important to avoid array operations as much as possible, as they only further increase the cost when potentially cheaper workarounds might be available.

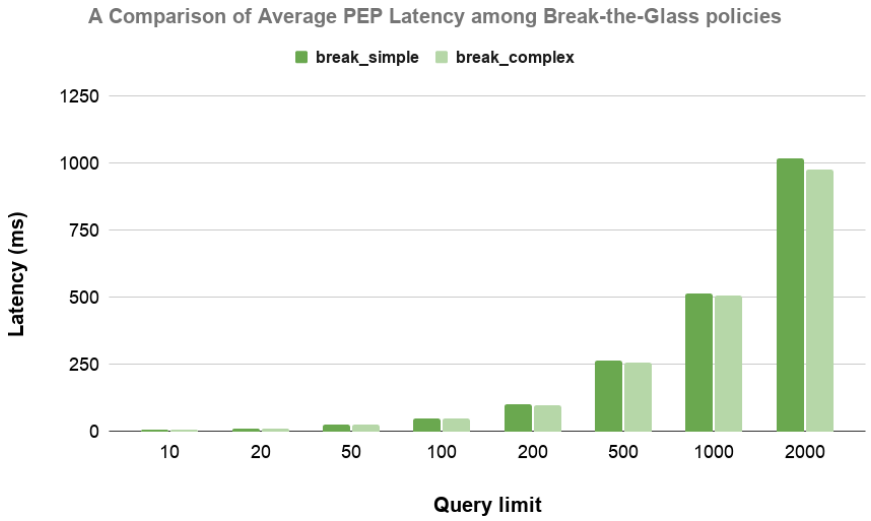

Figure 16: Average PEP Latency in Break-the-Glass category

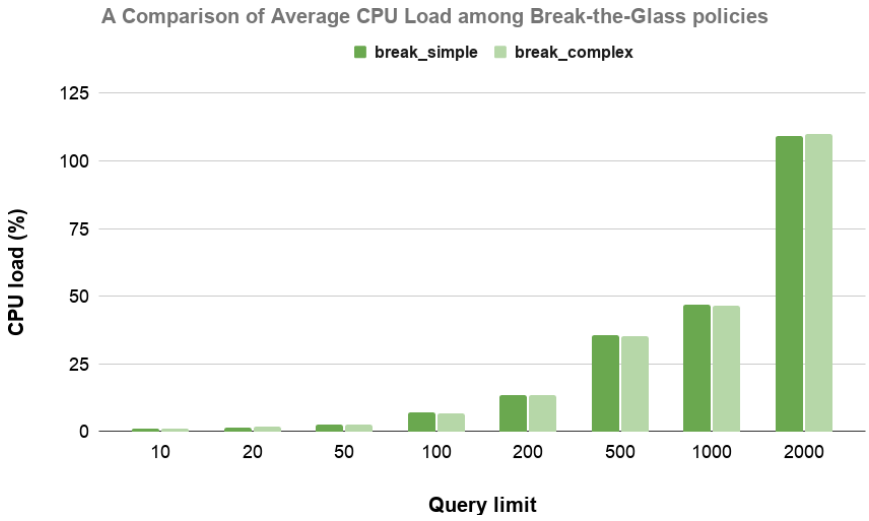

Figure 17: Average CPU load in Break-the-Glass category

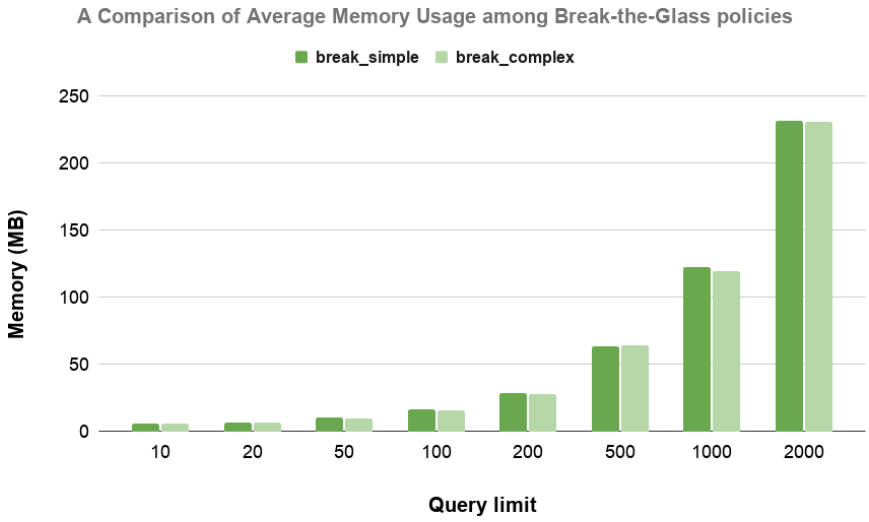

Figure 18: Average Memory usage in Break-the-Glass category

4.3 PEP Performance in Infrastructure

It is interesting to see how the latency of the PEP node affects the latency of the entire infrastructure. From each policy pair we took the one with the greater response delay and compared it with the system latency in Figures 19, 20, 21, and 22.

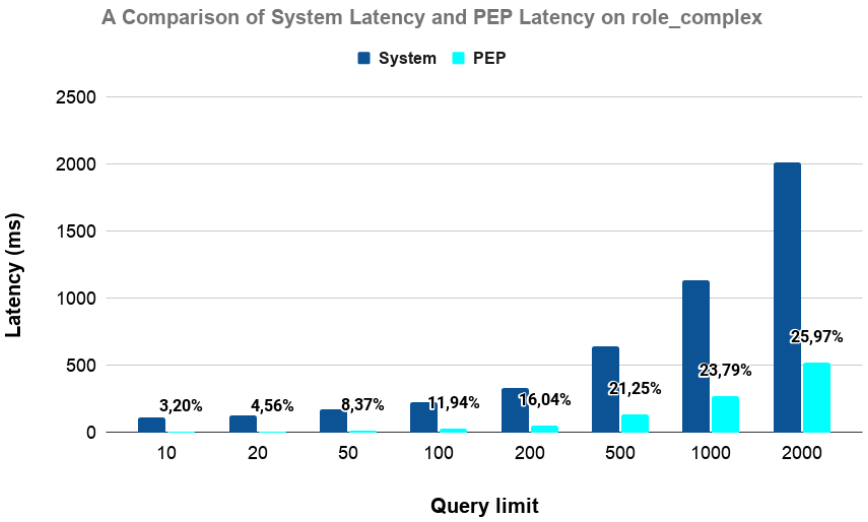

Figure 19: System Latency and PEP Latency on role_complex

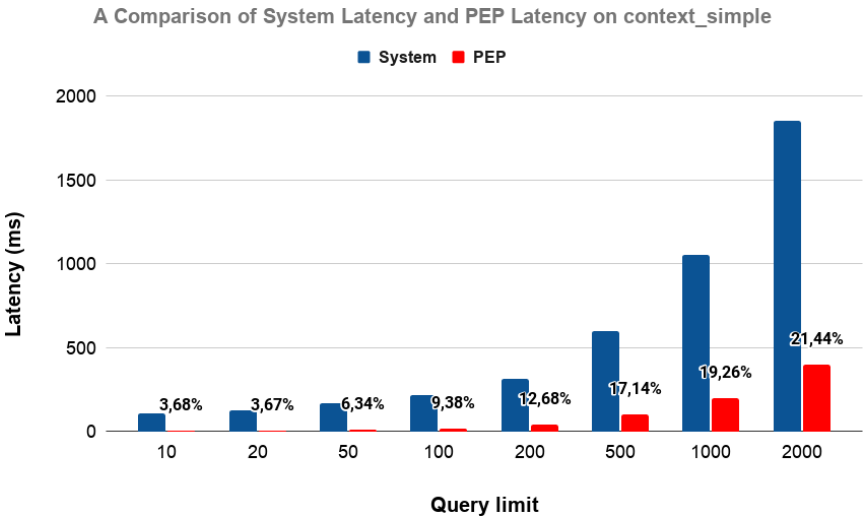

Figure 20: System Latency and PEP Latency on context1

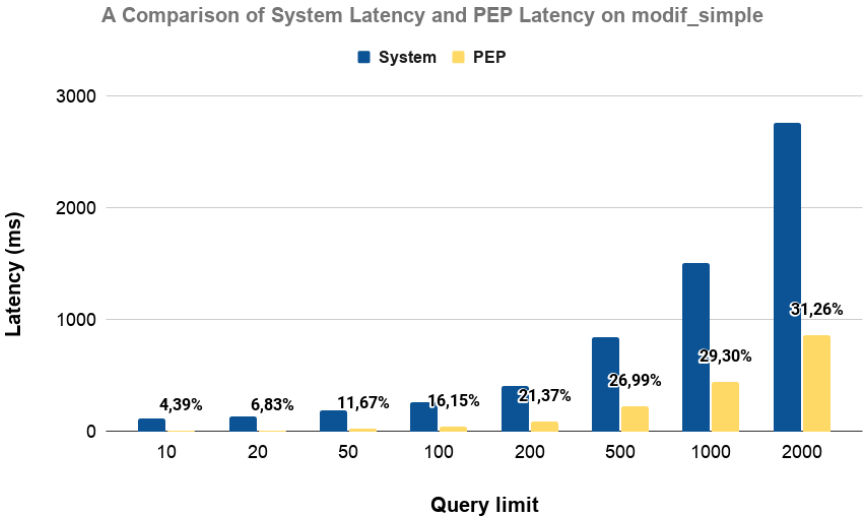

Figure 21: System Latency and PEP Latency on modif_simple

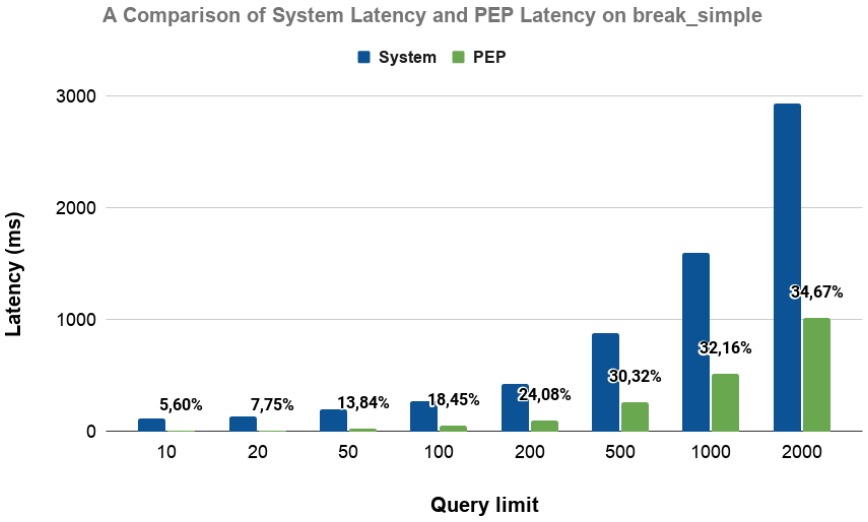

Figure 22: System Latency and PEP Latency on *break_simple*

Based on these results, we may conclude that when the data set is increased, the increase in system-wide latency, PEP latency and the relative latency of the two components are all nonlinear.

The difference between the complexity of the various policy categories, on the other hand, is not always as clear as we initially assumed. The results of *role_complex* and *context_simple*, for example, are almost identical, and the care team identifying *role_complex* even turns out to be somewhat more demanding than context1, which has to iterate and filter the contents of an embedded array, and with an input of size 2000 on *context_simple*, the PEP only provides the 21.4370% of the full system latency and 25.9672% on *role_complex*.

However, most of our expectations were confirmed by the results we obtained. Although it is clear that the identification of good practices, patterns and anti-patterns is necessary in the next phase of our research to further optimize the use of the PEP, the relative complexity and cost of the different categories were as expected. There is an overall latency and efficiency of the prototype infrastructure – a barely noticeable increase when we consider that the majority of healthcare applications, including our client application in its unmodified state. This requests 100 or 200 documents in a single operation, and we found the PEP (and OPA as its implementation) to be a very effective component of our security solution.

5 Conclusions

On the basis of our results, we conclude that the PEP (OPA) is indeed a suitable choice for our architecture and that the concept can be further developed. In the previous sections, we presented our proposed infrastructure and methodology, the categorization of security policies required in the healthcare sector, and a set of policies for different evaluations. When interpreting the results, we demonstrated the effectiveness of our concept, with only a relatively small increase in the overall latency in exchange for an effective healthcare security solution that is independent of the exact back- and front-end applications. In addition, we have also identified some good or definitely avoidable practices in the OPA policy definition that require further research to make it feasible in practice. To summarize these results based on the requirements described in the Introduction for an appropriate policy enforcement element in the domain of telemedicine:

- We showed that our proposed PEP can function and evaluate access in a telemedicine infrastructure with a minimal impact on the overall latency of the system, without any component explicitly being aware of its presence. Even with the largest input size, the evaluation delay caused by its inclusion was at most 35% of the total response delay;

- We showed that it can evaluate and transform the input with manageable CPU load and memory usage even for inputs that are unrealistically large in the healthcare workflow;

- Since our PEP configuration only depends on the input structure and our current selection is also available in Go and WebAssembly languages, it can be effectively used at any point in the infrastructure. Because of its internal database it can also work when the upper levels of the infrastructure are unavailable.

- We shoved that every category of our policy terminology can be effectively implemented in the script language of the OPA engine.

In the future we plan to continue our research in several directions. When interpreting the results, we found several possible patterns and anti-patterns for writing PEP policies. We intend to investigate these further in simulated an real-world scenarios as well, identify the possible bottlenecks and bad practices for deployment and provide other developers and researchers with a good guideline for implementing healthcare policies.

Since our infrastructure prototype turned out to be successful, the next step will be to further evaluate the capabilities of our proposed PEP integration in terms of portability, place it at different points in the infrastructure and record their behavior and efficiency.

If we succeed with these future steps, we hope to be a step closer to solving the problem of security and access control in telemedicine.

References

[1] Index – FHIR v4.0.1. `https://www.hl7.org/fhir/`. (Accessed on 09/16/2020).

[2] MongoDB official documentation. `https://docs.mongodb.com/manual/`. (Accessed on 09/16/2020).

[3] Open Policy Agent – policy performance. `https://www.openpolicyagent.org/docs/latest/policy-performance/`. (Accessed on 09/16/2020).

[4] Open Policy Agent official site. `https://www.openpolicyagent.org/`. (Accessed on 09/16/2020).

[5] Alhaqbani, Bandar and Fidge, Colin. Access control requirements for processing electronic health records. In *International Conference on Business Process Management*, pages 371–382. Springer, 2007. DOI: `10.1007/978-3-540-78238-4_38`.

[6] Alnefaie, Seham, Cherif, Asma, and Alshehri, Suhair. Towards a distributed access control model for IoT in healthcare. In *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–6. IEEE, 2019. DOI: `10.1109/CAIS.2019.8769462`.

[7] Altamimi, Ahmad. SecFHIR: A security specification model for fast healthcare interoperability resources. *International Journal of Advanced Computer Science and Applications*, 7(6), 2016. DOI: 10.14569/IJACSA.2016.070645.

[8] Andrew, Marcus. Security in FHIR at DevDays Redmond 2019. https://tinyurl.com/ryk9zlu, 2019. (Accessed on 07/20/2020).

[9] Attia, Hasiba Ben, Kahloul, Laid, and Benharzallah, Saber. A new hybrid access control model for security policies in multimodal applications environments. *Journal of Universal Computer Science*, 24(4):392–416, 2018. DOI: 10.3217/jucs-024-04-0392.

[10] Carter, Gracie, Chevellereau, Ben, Shahriar, Hossain, and Sneha, Sweta. OpenPharma Blockchain on FHIR: An interoperable solution for read-only health records exchange through blockchain and biometrics. *Blockchain in Healthcare Today*, 2020. DOI: 10.30953/bhty.v3.120.

[11] Chaballout, Basil Harris, Shaw, Ryan Jeffry, and Reuter-Rice, Karin. The SMART healthcare solution. *Advances in Precision Medicine*, 2(1), 2017. DOI: 10.18063/apm.v2i1.213.

[12] Conley, Ed and Pocs, Matthias. GDPR compliance challenges for interoperable health information exchanges (HIEs) and trustworthy research environments (TREs). *European Journal of Biomedical Informatics*, 14(3), 2018. DOI: 10.24105/ejbi.2018.14.3.7.

[13] Fernández-Alemán, José Luis, Señor, Inmaculada Carrión, Lozoya, Pedro Ángel Oliver, and Toval, Ambrosio. Security and privacy in electronic health records: A systematic literature review. *Journal of Biomedical Informatics*, 46(3):541–562, 2013. DOI: 10.1016/j.jbi.2012.12.003.

[14] Ferraiolo, David, Chandramouli, Ramaswamy, Kuhn, Rick, and Hu, Vincent. Extensible access control markup language (XACML) and next generation access control (NGAC). In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, pages 13–24, 2016. DOI: 10.1145/2875491.2875496.

[15] Ferraiolo, David, Cugini, Janet, and Kuhn, D Richard. Role-Based Access Control (RBAC): Features and motivations. In *Proceedings of 11th Annual Computer security Application Conference*, pages 241–48, 1995.

[16] Finance, Beatrice, Medjdoub, Saida, and Pucheral, Philippe. Privacy of medical records: From law principles to practice. In *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)*, pages 220–225. IEEE, 2005. DOI: 10.1109/CBMS.2005.89.

[17] Gajanayake, Randike, Iannella, Renato, and Sahama, Tony R. Privacy oriented access control for electronic health records. In *Data Usage Management on the Web Workshop at the Worldwide Web Conference*. ACM, 2012.

[18] Islam, SM Riazul, Kwak, Daehan, Kabir, MD Humaun, Hossain, Mahmud, and Kwak, Kyung-Sup. The Internet of Things for health care: A comprehensive survey. *IEEE access*, 3:678–708, 2015. DOI: 10.1109/ACCESS.2015.2437951.

[19] Johnson, Alistair EW, Pollard, Tom J, Shen, Lu, Li-Wei, H Lehman, Feng, Mengling, Ghassemi, Mohammad, Moody, Benjamin, Szolovits, Peter, Celi, Leo Anthony, and Mark, Roger G. MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3(1):1–9, 2016. DOI: 10.1038/sdata.2016.35.

[20] Joshi, Maithilee, Joshi, Karuna, and Finin, Tim. Attribute based encryption for secure access to cloud based EHR systems. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 932–935. IEEE, 2018. DOI: 10.1109/CLOUD.2018.00139.

[21] Mukherjee, Subhojeet, Ray, Indrakshi, Ray, Indrajit, Shirazi, Hossein, Ong, Toan, and Kahn, Michael G. Attribute based access control for healthcare resources. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pages 29–40, 2017. DOI: 10.1145/3041048.3041055.

[22] Pal, Shantanu, Hitchens, Michael, Varadharajan, Vijay, and Rabehaja, Tahiry. Fine-grained access control for smart healthcare systems in the Internet of Things. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, 4(13), 2018. DOI: 10.4108/eai.20-3-2018.154370.

[23] Peleg, Mor, Beimel, Dizza, Dori, Dov, and Denekamp, Yaron. Situation-based access control: Privacy management via modeling of patient data access scenarios. *Journal of Biomedical Informatics*, 41(6):1028–1040, 2008. DOI: 10.1016/j.jbi.2008.03.014.

[24] Premarathne, Uthpala, Abuadbba, Alsharif, Alabdulatif, Abdulatif, Khalil, Ibrahim, Tari, Zahir, Zomaya, Albert, and Buyya, Rajkumar. Hybrid cryptographic access control for cloud-based EHR systems. *IEEE Cloud Computing*, 3(4):58–64, 2016. DOI: 10.1109/MCC.2016.76.

[25] Pussewalage, Harsha S Gardiyawasam and Oleshchuk, Vladimir A. An attribute based access control scheme for secure sharing of electronic health records. In *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*, pages 1–6. IEEE, 2016. DOI: 10.1109/HealthCom.2016.7749516.

[26] Ray, Indrakshi, Alangot, Bithin, Nair, Shilpa, and Achuthan, Krishnashree. Using attribute-based access control for remote healthcare monitoring. In *2017 Fourth International Conference on Software Defined Systems (SDS)*, pages 137–142. IEEE, 2017. DOI: 10.1109/SDS.2017.7939154.

[27] Ray, Indrakshi, Ong, Toan C, Ray, Indrajit, and Kahn, Michael G. Applying attribute based access control for privacy preserving health data disclosure. In *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 1–4. IEEE, 2016. DOI: 10.1109/BHI.2016.7455820.

[28] Rezaeibagha, Fatemeh and Mu, Yi. Distributed clinical data sharing via dynamic access-control policy transformation. *International Journal of Medical Informatics*, 89:25–31, 2016. DOI: 10.1016/j.ijmedinf.2016.02.002.

[29] Rosa, Marco, Faria, Cristiano, Barbosa, Ana Madalena, Caravau, Hilma, Rosa, Ana Filipa, and Rocha, Nelson Pacheco. A fast healthcare interoperability resources (FHIR) implementation integrating complex security mechanisms. *Procedia Computer Science*, 164:524–531, 2019. DOI: 10.1016/j.procs.2019.12.215.

[30] Rousskov, Alex and Soloviev, Valery. A performance study of the Squid proxy on HTTP/1.0. *World Wide Web*, 2(1-2):47–67, 1999. DOI: 10.1023/A:1019240520661.

[31] Sánchez, Yaira K Rivera, Demurjian, Steven A, and Baihan, Mohammed S. Achieving RBAC on RESTful APIs for mobile apps using FHIR. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 139–144. IEEE, 2017. DOI: 10.1109/MobileCloud.2017.22.

[32] Sun, Jianfei, Xiong, Hu, Zhang, Hao, and Peng, Li. Mobile access and flexible search over encrypted cloud data in heterogeneous systems. *Information Sciences*, 507:1–15, 2020. DOI: 10.1016/j.ins.2019.08.026.

[33] Szabó, Z. and Bilicki, V. Felhőben tárolt egészségügyi adatok védelme ABAC modellel. In *Orvosi informatika 2018 — A XXXI. Neumann Kollokvium konferencia-kiadványa*, pages 134–139. Neumann János Számítógép-tudományi Társaság (NJSZT), 2018.

[34] Szabó, Z. and Bilicki, V. Achieving RBAC on RESTful APIs for mobile apps using FHIR. In *CSCS — The Twelfth Conference of PhD Students in Computer Science*. Institute of Informatics, University of Szeged, 2020. https://www.inf.u-szeged.hu/~cscs/proceedings.php.

[35] Tasali, Qais, Chowdhury, Chandan, and Vasserman, Eugene Y. A flexible authorization architecture for systems of interoperable medical devices. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, pages 9–20, 2017. DOI: 10.1145/3078861.3078862.

[36] Yuan, Eric and Tong, Jin. Attributed based access control (ABAC) for web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005. DOI: 10.1109/ICWS.2005.25.