SmartAccess: Attribute-Based Access Control System for Medical Records Based on Smart Contracts

MARCELA TULER DE OLIVEIRA, LÚCIO HENRIK AMORIM REIS, YIANNIS VERGINADIS, DIOGO MENEZES FERRAZANI MATTO, AND SÍLVIA DELGADO OLABARRIAGA

1Epidemiology and Data Science Department, Amsterdam UMC Location University of Amsterdam, 1105 AZ Amsterdam, The Netherlands
2Biomedical Engineering and Physics Department, Amsterdam UMC Location University of Amsterdam, 1105 AZ Amsterdam, The Netherlands
3MídiaCom—TET/PPGEET/UFF, Universidade Federal Fluminense, Niterói, Rio de Janeiro 24020-140, Brazil
4Department of Business Administration, Athens University of Economics and Business (AUEB), 104 34 Athens, Greece
5Institute of Communications and Computer Systems, National Technical University of Athens (NTUA), 157 80 Athens, Greece

Corresponding author: Marcela Tuler de Oliveira (m.tuler@amsterdamumc.nl)

This work was supported by ASCLEPIOS: Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare (European (EU) Research Project) under Project 826093.

ABSTRACT Cross-organisation data sharing is challenging because all the involved organisations must agree on ‘how’ and ‘why’ the data is processed. Due to a lack of transparency, the organisations need to trust that others comply with the agreements and regulations. We propose to exploit blockchain and smart contracts technologies to define an Attribute-Based Access Control System for cross-organisation medical records sharing, coined SmartAccess. SmartAccess offers joint agreement over access policies and dynamic access control besides blockchain transparency and auditability. We leverage the Attribute-Based Access Control model to implement smart contracts. We deploy and test them on a private and permissioned blockchain, transforming the access control process into a distributed smart contract execution. This paper proposes the SmartAccess system and its application in two healthcare use cases. We introduce the threat model and perform a security analysis of the system. To demonstrate the feasibility of our proposal, we implement a proof-of-concept of the smart contracts, written in Solidity language, with a size-efficient policy representation, and analyse the complexity and scalability of the contracts’ functions. Furthermore, we present performance results, measuring the latency and throughput of the transactions to execute the access control functions with different blockchain network consensus setups. We also compare the performance of the SmartAccess system against two open-source Solidity implementations of smart contract-based access control, Role-based Access Control and Access Control List. Finally, we discuss the strengths and drawbacks of our proposal. SmartAccess requires the overhead of a decentralised system, but the trade-off is transparency, regulation compliance and auditability for complex cross-organisation data sharing.

INDEX TERMS Attribute-based access control, blockchain, cross-organisation security, electronic medical records, GDPR, healthcare information system, smart contracts.

I. INTRODUCTION

In the last decades, organisations have increasingly processed personal data; however, ensuring that data are leveraged only for legitimate purposes remains a significant challenge. Organisations must implement adequate access control mechanisms to safeguard legitimate data access when handling personal data, particularly when sharing data across organisations. Although there is plenty of data and value to be exploited, various data sharing barriers exist, including legal
data protection regulations, which might differ across the various organisations.

According to the European General Data Protection Regulation (GDPR) [1], “Personal data means any information relating to a data subject, an identified or identifiable natural person”. Besides the data subject, the GDPR also defines the two prominent roles and responsibilities of data processing: the data controller is “who determines the purposes for which and how personal data is processed” and the data processor is “who processes personal data only on behalf of the controller” [2]. Moreover, to support cross-organisation data processing, the GDPR also defines joint controllers who “together with one or more organisations jointly determine ‘why’ and ‘how’ personal data should be processed” [3]. The organisations usually agree on the terms of data sharing and sign a legally-binding document, coined the data processing agreement (DPA) [4]. Unfortunately, the compelling practical and ethical justifications usually defined in the DPA for the ‘why’ are not universally understood by all parts. Moreover, the auditing process to check compliance with the DPA relies on information technology applications records that usually fail to inform about ‘why’ and ‘how’ the personal data has been processed transparently.

Healthcare is a typical case of cross-organisational data sharing because healthcare professionals from multiple organisations need to process patients’ data to perform their tasks. The patient’s data here means the electronic medical records (EMR), which contain personal data about the patient. The patient’s data are usually spread among hospitals and clinics that have treated the patient at least once. Thus, each organisation holds only some part of the data. In many cases, parts of the data remain unreachable, even in emergencies, because they are located in information systems outside the treating organisation’s boundaries. This limitation could be addressed by using shared medical record systems that exploit cloud solutions [5], [6], [7] or distributed database systems, such as the InterPlanetary File System (IPFS) [8]. These shared systems promise to provide the required data consistency and availability for healthcare purposes. The healthcare organisations then act as joint controllers, defining and enforcing strict cross-organisation access control to patients’ data. A successful access control system must be dynamic and granular to support the complex nature of cross-organisational data sharing in healthcare. For example, a doctor must have access to a patient’s data during treatment, but the access must be revoked when the doctor finishes the care. Moreover, all the access logs should be available for the audit process and regulatory compliance.

The Attribute-Based Access Control (ABAC) model offers a dynamic and fine-grained access control approach to protect personal data [9]. ABAC defines policies as combinations of rules and attributes that can be as granular as necessary. ABAC also uses context expression and contextual attributes, which gives dynamism to the policy evaluation. Thus, ABAC defines not only “why” and “how” but also “by whom”, “when” and “where” the personal data can be processed. A drawback of current systems that use the ABAC model is that it usually delegates access control management to the data storage system administrator, where the policies are defined, managed, evaluated, and enforced. Hence, the data controllers and subjects must trust that the storage system administrator will follow the defined policies and not allow any processing of personal data that does not comply with the respective policies.

Regarding transparency, the data controllers also depend on the data storage systems to keep and disclose records about data processing activity on personal data. Recently, blockchain and smart contracts concepts have been proposed to facilitate transparency over data access control [10], [11], [12]. Other researchers proposed using blockchain to securely achieve data provenance and auditing while sharing medical data among other cloud service providers [13]. Others proposed a blockchain-based medical data marketplace in which medical record sellers can sell their data to interested buyers [14]. In these proposals, the data processing is logged as transactions of the blockchain, which are immutable and transparent for auditing. However, to the best of our knowledge, none of the proposals considers the complexity and dynamism required for healthcare cross-organisation data sharing. In most proposals, the so-called ‘data owner’ defines the access control policies, but the ‘data owner’ of the patient’s data is an unclear role [15] and it is not defined in the GDPR. Moreover, relying on the patients to define specific access control policies for their data is unrealistic.

Our research focuses on exploiting smart contracts and blockchain technology through the ABAC model for dynamic access control of personal data across organisations. Thus, we must overcome the following challenges: First, healthcare organisations should agree and comply with common access control policies for patient data processing as joint data controllers. Second, the access control system should guarantee a valid purpose for data processing. Third, policy decisions and enforcement should not depend on a centralised trustworthy party. Finally, the logs of data access activity should be transparent and auditable.

The paper proposes the SmartAccess system that follows the reference architecture defined by the XACML standard [16] and implements the ABAC components as smart contracts. In the blockchain network, the data controllers jointly define and manage the policies, in consensus, with the rest of the data controllers in the blockchain network. The data processors and the context provide attributes validated by their data controllers to justify and legitimate the data processing activities. Data processing is only allowed if the data processor runs the smart contracts and has the right attributes to comply with the policy’s rules, therefore, yielding a permit decision. Data processors run the access control smart contracts locally on their computers without relying on a central party. Every function executed in the smart contracts generates an auditable transaction published in the blockchain. Therefore, any node of the network can always search for the access requests that have been performed to
given data, the values of the attributes at that time, the policy enforced during the access evaluation, and the resulting access decision. Moreover, we analyse the security threats of the SmartAccess system and demonstrate the proposed solution in different use cases healthcare scenarios: access with patient consent and access during acute care. Finally, we demonstrate the feasibility of the solution through the implementation of a size-efficient policy representation and experiments that evaluate the complexity and scalability of the smart contract functions. Moreover, we test the performance of the SmartAccess system in different blockchain network setups, measuring the latency and throughput of the transaction. Additionally, we compare the performance of SmartAccess with two Solidity implementations of the Role-Based Access Control (RBAC) and Access Control List (ACL) mechanisms.

II. BACKGROUND AND RELATED WORK
This section presents the background of the concepts adopted in the SmartAccess system, namely blockchain, smart contracts and the Attribute-Based Access Control Model. We also present related work and summarise their characteristics in a table.

A. BLOCKCHAIN AND SMART CONTRACTS
Blockchain technology is defined by two essential elements: the data structure of the blocks and the peer-to-peer network composed of the participant nodes [17]. Blockchain allows running distributed applications without needing a trusted third party while meeting security requirements such as integrity, authenticity, non-repudiation, and accountability [18].

The blockchain data structure comprises blocks concatenated to the previous one via hash values. Except for the first (genesis block), each subsequent block contains valid transactions and the hash of the content of the previous block. Hence, the hash concatenation between the blocks ensures the integrity and immutability of the transactions stored in each block since these cannot be tampered with without breaking the hash chaining. The integrity and immutability of the transactions rely on the peer-to-peer network. Each node of the network has a copy of the hash chaining, and because of the network consensus mechanism, all the nodes have the same global view of the blockchain. Nodes can order and package validated transactions into a candidate block to be inserted into the blockchain. The successful insertion of a block in the chain is known as ‘mining’, and the nodes eligible to insert blocks in the blockchain are the ‘miners’. The selection of miners depends on the consensus mechanism [19].

The adopted consensus mechanism depends on the type of network and the roles of the nodes. The blockchain network can be private or public, and the nodes’ roles can be permissioned or permissionless [20]. In a public blockchain, everyone can join and leave the network; on the other hand, in a private blockchain, the nodes must be added to the blockchain and known by the entire network. In a permissionless network, all nodes play the same role, i.e., every node in the network can mine blocks and participate in the consensus. The network is called permissioned when the nodes are divided into groups according to their role, e.g., miner nodes and nodes that only generate transactions. Usually, in a permissioned blockchain, the role of a node is assigned when the node joins the network [21].

Blockchain was initially proposed to support financial transactions of virtual cryptocurrency in the Bitcoin network [17]. A transaction is a message sent from one node address to another. It can include binary data, which is called ‘payload’. The technology has evolved, and it currently can be used to implement more complex transactions, known as smart contracts, first implemented in the Ethereum network [18]. Smart contract is a code with executable functions and storage space for its data, of which all nodes have a replica locally.

When a smart contract is deployed, the contract owner broadcasts a transaction carrying the payload as a code linked to a public address. Once this transaction is mined, all nodes store a replica of the smart contract [22]. To execute a smart contract function, the ‘sender node’ broadcasts a transaction to the network with the smart contract’s public address, carrying the function input arguments inside the transaction payload. The sender node then waits for the transaction to be validated and mined in a block. Meanwhile, from the point of view of all other nodes, the transaction has not yet happened. Once the transaction is mined and broadcast to the network, the transaction is considered executed. The miner node sends a ‘transaction receipt’ to the address of the sender node, confirming that the transaction was mined. Therefore, the ‘event’ is emitted at that point. It means that the sender node executes the smart contract function with the input arguments of the transaction and emits the event that the user front-end can then process [23]. Although the function is executed locally, any node can verify the event transactions executing the smart contracts with the same input arguments. A smart contract function can also execute functions from other contracts as a process. In order to send transactions and execute functions, it may be required for the sender to pay a fee. In Ethereum, the fee is referred to as ‘gas’. In Ethereum Virtual Machine (EVM) based blockchain, gas is an excellent way to evaluate the complexity of the smart contracts since the gas usage increases with the complexity of the transactions. The smart contracts’ global state can be seen as a virtual machine running all the code on the blockchain.

B. ATTRIBUTE-BASED ACCESS CONTROL MODEL
The ABAC model defines an access control paradigm by which access rights are granted to the data requester by using policies that consist of logical combinations of attributes. ABAC policies, requests and responses are expressed in the XACML language, an OASIS [24] standard. A policy is a combination of rules that the requester must obey. We achieve this by combining algorithms at the policy set level (i.e. policy combining algorithms) or at the policy level...
(i.e. rule combining algorithms). Each algorithm defines how to properly merge the evaluation of the different requests to produce a unique decision. The policies are associated with the ‘targets’, which can be a resource, a type of action, a context expression, or a combination of these. A context expression describes the circumstances under which access should be allowed. For example, a policy protects a combination of targets: a patient’s data (resource) to be read (type of action) in case of an emergency (context expression). When a request is issued, the rules expressed in the policies are evaluated, exploiting the attribute values to return a response. The responses contain the decision concerning the request. A response can be ‘Permit’, ‘Deny’, ‘Indeterminate’ (in case of errors or missing values) or ‘Not applicable’ (the request does not regard any of the policies).

The XACML standard defines five main components that handle access decisions, namely the Policy Administration Point (PAP), Policy Enforcement Point (PEP), Policy Decision Point (PDP), Policy Information Point (PIP), and Context Handler (CH). The PAP stores and manages a persistent pool of policies associated with the target identifiers. The PEP constitutes the integration hook to any system, where the resources to be secured are stored and managed. A PEP receives the access requests and freezes the execution workflow until a decision is yielded. At the same time, it propagates the requests to the PDP, which is the core decision place for any incoming access request. The PDP retrieves all the necessary attributes and contextual information from the PIP, evaluates the defined policies, and yields a decision accordingly. The PIP is responsible for retrieving and storing attribute values. The Context Handler (CH) is responsible for deriving the context of a certain request and, in some recent efforts [9] for semantically uplifting the attribute values stored in the PIP to infer additional context.

### C. RELATED WORK

Here we discuss the most relevant literature proposing solutions for access control to medical data using blockchain technology. In Table 1, we classify the related work according to XACML’s five main components. Even if the access control proposed in the paper is not based on the XACML standard, we evaluate if the proposed solution uses blockchain to handle each part of the access control process as defined in XACML. We mark with ✓ in the respective column when a solution uses the blockchain to store or manage the access control policies (PAP), to handle the requests and deliver the decision (PEP), to evaluate the policies (PDP), to store and manage the attributes used to evaluate the policies (PIP) and to store and manage contextual attributes used to evaluate the policies (CH). Moreover, we also mark with ✓ if an implementation is presented in the papers. Below, we present a summary of the proposed solutions and how SmartAccess differs from existing literature.

According to Dias et al. [25], current systems that attempt to share access control policies between healthcare entities are prone to system and network faults and do not assure the integrity of policies. The authors propose using a consortium blockchain, where the ABAC policies are stored off-chain, the pointers are stored as transactions, and the different entities know all the parties that can act over the e-Health resources. Although the approach allows entities to maintain a consensus about the policies, it is limited to sharing policies. SmartAccess goes beyond and uses smart contracts technology to manage, evaluate and enforce access control policies in a distributed system.

Fan et al. [26] propose Medblock, a data-sharing framework with an access control mechanism based on a signature scheme. The sensitive data and the pointers to the patient EMR are encrypted with the multi-signature scheme inside the blockchain. The access control mechanism traverses the blocks until it finds the right block by comparing the signature with the signature collection on the ledger. Whether the user can see the encrypted content on the block depends on the comparison result. Zhang et al. [27] propose the FHIRChain for data sharing among clinicians and researchers based on the FHIR standard. FHIRChain addresses five key interoperability requirements: user identifiability and authentication, secure data exchange, permissioned data access, consistent data formats, and system modularity. The data access control is based on a smart contract that results in an access token. The access tokens are defined for each data transaction, which uses asymmetrical encryption to protect the data pointers off-chain. This design uses the users’ digital health identities to encrypt content so that only users holding the correct digital identity private keys can decrypt the content. Daraghmi et al. [28] propose MedChain, an incentive consensus mechanism that leverages the degree of the reputation of health providers regarding their efforts in maintaining medical records and creating new blocks. The access control contract includes all permissions-related information specific to every record based on smart contracts. It lists the Ethereum addresses for all users who have access permissions on the record. This contract specifies the level of access and the symmetric key encrypted with each user’s public key. Although [26], [27], [28] promise an effective access control, none offer dynamic access control for daily healthcare needs.

| Ref | PAP | PEP | PDP | PIP | CH | Implementation |
|-----|-----|-----|-----|-----|----|----------------|
| [25] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| [26] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| [27] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| [28] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| [29] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| [30] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| [31] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| [32] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| [10] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| [11] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| [12] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| This work | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
In those proposals, the access permissions need to be listed or specified beforehand the need for data.

Azaria et al. [29] propose MedRec, a decentralised EMRs management system using blockchain technology. MedRec is a modular design that manages permissions, authorisation, and data sharing between participants. The authors highlight the ability of MedRec to encrypt outside data and preserve hash pointers to patients’ health records along with their access permissions in the blockchain. Like MedRec, Dagher et al. [30] propose Ancile, an Ethereum-based blockchain for a record management system that utilises smart contracts for heightened access control and data obfuscation. Ancile keeps the patients’ medical records in the existing databases of providers, and reference addresses to these records and their permissions to each record are stored in the smart contract. It was designed to store the Ethereum addresses of all nodes that may interact with a record, a level of access, and a symmetric key encrypted with each node’s public key. Dubovitskaya et al. [31] propose an EHR system where the patients specify the access control policies and permissions to each healthcare professional. Rajput et al. [32] propose a system that allows the ‘owner’ of the records to assign the rules for an emergency team or staff member who can obtain permission to access the current information from the resource by considering the time restriction. The access control is based on a list of pre-registered emergency teams and physicians.

Rauhani et al. [11] proposed an ABAC system for EMR data sharing. It is different from ours because it does not consider context expression, which is fundamental for healthcare applications where access rights are dynamic and depend on the patient’s needs. Maesa et al. [10] also proposed an ABAC system using blockchain. However, the values of attributes cannot be changed since they are stored on the blockchain. They are auditible since their updates can be executed only through blockchain transactions, which are recorded on the blockchain. Both proposals [10], [11] do not consider that the attributes must be authenticated by the organisations where the processor works every time they interact with the access control system. As an asynchronous system, the blockchain would require that the organisations update the attributes of their professionals on the blockchain continuously, which would be a burden for the dynamic attributes of the healthcare professionals. On the other hand, [12] proposed to keep the user attributes out of the blockchain and rely on trusted authorities to maintain a list of users associated with their verified attributes. Then using a smart contract, these authorities authenticate the user’s attributes during the user request for data. SmartAccess uses an authentication token as the source of the attributes for professionals and the environment. Thus, every time the professional performs a request, the professional must provide an access token signed by the organisation.

The proposals [25], [27], [29], [32] use PoW and incentivise health providers and medical researchers to participate in mining by earning an Ether, an Ethereum-based currency unit, to fund their activities’ continuation. In other words, they participate in mining to get beneficiaries from the network, although most of the current healthcare systems are welfare-oriented with no intent to involve any monetary value. Because of that, SmartAccess uses non-monetary consensus mechanisms based on voting, Byzantine Fault Tolerance (BFT) consensus [33]. Furthermore, all the proposals [29], [30], [31] based on a pre-defined list of users cannot be used in emergency situations since a patient in an emergency cannot know the specific teams or professionals that will treat them. Moreover, when the authors refer to ‘data owners’, they are referring to ‘patients’, the data subjects who usually have no control over the data collected about them. Although these proposals give more control to the data subject, in the healthcare scenario, according to GDPR, healthcare organisations are responsible for access control to the EMR.

III. SmartAccess: ATTRIBUTE-BASED ACCESS CONTROL BASED ON SMART CONTRACTS

This section proposes SmartAccess, a new blockchain-based access control system to secure personal data across organisations. First, we present an overview and a description of the network and smart contracts. Then, we describe the access control flow, showing how the network nodes interact with the smart contracts.

A. OVERVIEW

SmartAccess is based on a blockchain peer-to-peer network and smart contracts. The network nodes use smart contracts to define access control policies, request access permission and verify whether the request is permitted or not. Following the XACML, we denote ‘resource’ as any data that can be requested using SmartAccess and ‘target’ as the combination of context expression, action and resource requested. Moreover, any network node can audit the requests, targets, decisions, and policy enforcement logs that are transparent in the blockchain.

We assume that nodes run user applications that would enable them to be authenticated, search for the resource identifiers and request for the resource stored off-chain. The actual user application is considered out of the scope of this paper because it would be specific to a use case and perhaps to a blockchain framework. Moreover, we assume that the usage of cryptography to protect resources is necessary, but it is out of the scope of this research. In the future, we envision that SmartAccess would be used as an access control layer above any cryptographic layers used to protect the resource, for example, for protecting both the encrypted data and the respective cryptographic keys. Therefore, here we focus on the description of a generic access control mechanism based on smart contracts in a distributed and transparent manner.

Fig. 1 shows an overview of the network and the smart contracts. The network comprises data controller, storage and processor nodes, which have different roles according to their function regarding the GDPR. The smart contracts running...
FIGURE 1. Overview of SmartAccess, which is composed of a peer-to-peer blockchain network composed of controller, processor and storage nodes, and four smart contracts: Enforcement Smart Contract (PEPSC), Decision Smart Contract (PDPSC), Policies Smart Contract (PAPSC) and Information Smart Contract (PIPSC).

The proposed blockchain network is private and permissioned, where nodes are added and removed from the network and have roles and permissions. A node is a connected device or server from where the user can send transactions, execute smart contracts and store the blockchain. A node also runs an application implementing the business logic layer. Every node has a pair of asymmetric keys used to generate its address and digital signature. Below, we describe how each node type maps to the GDPR roles and explain their responsibilities in the blockchain network.

Controller Node ($n_c$): represents an organisation with the data controller role in GDPR. Each organisation runs its own node, which participates in the consensus mechanism, generates blocks, and manages the network and the smart contracts. Controller nodes define and deploy access control policies and validate the proposed policies through consensus. It can also add new controllers, processors and storage nodes, as well as manage the attributes of their processor nodes, which are the professionals that work for the organisation. Controller nodes are also responsible for authenticating the processors and providing them with an authentication token that carries valid attributes for further interaction with the system. In a healthcare scenario, hospitals and clinics are examples of controllers that define policies and manage the access of professionals and patients to resources.

Processor Node ($n_p$): represents the data processor role in GDPR. A processor node is a requester in the ABAC model, and who runs the smart contracts to request access to a resource. In a healthcare scenario, the processor nodes are the healthcare professionals or the patients (data subjects). A data processor only processes the resource on behalf of the data controller. Each $n_p$ is added to the blockchain network by some $n_c$, and after that, they can receive requests from any $n_p$. When a request to access a resource is received, the storage node runs the smart contracts

Storage Node ($n_s$): represents a persistent data storage system, such as a cloud storage provider or IPFS, serving as a gateway between a resource and the $n_c$ and $n_p$. Storage nodes are added to the network by some $n_c$, and after that, they can receive requests from any $n_p$. When a request to access a resource is received, the storage node runs the smart contracts.
to verify if $n_p$ has permission to access the data. In healthcare, each healthcare organisation potentially has its own storage node, which contains parts of the patient EMR. Therefore, we assume that the storage nodes are part of an interoperable and decentralised data system.

### C. SMART CONTRACTS

The smart contracts (SC) implement the main components of the ABAC model according to XACML: PEPSC for policy enforcement, PDPSC for yielding decisions, PAPSC for policies management and PIPSC for attributes storage. The XACML Context Handler (CH) is distributed into the four SC because all requests and policies are driven by context expression. This means that SC handles the requests according to the circumstances and the purpose of the access and applies the appropriate policy.

Table 2 summarises SC and their functions (denoted <smart contract name>,<function name>). A sender is a node or smart contract allowed to emit the transaction to run the function. There is a checking condition inside each function to verify if the sender is allowed to run the function, and if the required condition is true, the node is allowed to run the function. The four SC contain the functions code and a relational database where each argument is saved and managed accordingly to each function execution. The functions can add, remove, load, change, and save a variable in the relational database of the smart contract. For example, the database contained in PAP stores the policies for each target in the following format (ContextExp, Actions, SubjectID), for example (‘emergency’, ‘read’, ‘123’). The result of the functions listed in the Table 3 describes what happens in the system when the function is successfully executed; otherwise, the execution stops without any change for the system.

Every transaction that executes a function is validated and mined in a new block added to the blockchain. The transaction payload logs the arguments inputted, such as the target, attributes, policies and decisions used when executing the corresponding function. Any network node can search for an argument and retrieve all the transactions containing the argument.

Below we describe the SC and their relationships with the XACML model. See Fig. 2 for a simplified overview.

1) **ENFORCEMENT SMART CONTRACT (PEPSC)**

The smart contract implements the PEP functionality in the ABAC XACML model. It contains four functions: PEPSC:RequestAccess to process the data access requests, and PEPSC:SaveDecision, PEPSC:RevokeAccess and PEPSC:VerifyDecision to manage the corresponding decisions. $n_p$ runs PEPSC RequestAccess, which triggers a call to PDPSC: EvaluateRequest. After the evaluation, the decision is taken, and PEPSC:SaveDecision is called back. PEPSC:RevokeAccess can be called by $n_p$, $n_c$, or by PIPSC:RevokeContextAttr when a contextual attribute is revoked. The PEPSC:VerifyDecision is called by $n_o$ to check if the requesting $n_p$ has valid access permission. Finally, the SaveRequestLogObligation called by the storage node to save the detailed attributes regarding the off-chain request, such as geolocation and origin IP address.

2) **DECISION SMART CONTRACT (PDPSC)**

The contract implements the PDP functionality in the ABAC XACML model, being the core decision point for any incoming access request. It contains only the PDPSC:EvaluateRequest function, which evaluates the access requests. This function is called by PEPSC:RequestAccess and requires the same input arguments. The evaluation process requires PDPSC to call functions in the other SC to acquire additional information: PAPSC:GetContextAttr and PEPSC:SaveDecision.

3) **POLICIES SMART CONTRACT (PAPSC)**

The contract implements the PAP functionality in the ABAC XACML model, where the policies are defined, managed and stored. In PAPSC, a Policy is related to the PolicyID, ContextExp, Actions, SubjectID and ResourceID. It contains four functions: PAPSC:CreatePolicy, PAPSC:ChangePolicy, PAPSC:RemovePolicy and PAPSC:LoadPolicy. Only $n_c$ may run the policy management functions of PAPSC to create, change or remove a policy. Also, only PDPSC calls PAPSC:LoadPolicy to retrieve a Policy for evaluation.

4) **INFORMATION SMART CONTRACT (PIPSC)**

The smart contract implements the PIP functionality in the ABAC XACML model, gathering the information necessary to evaluate the requests. PIPSC stores the addresses of all nodes. It contains seven functions: PIPSC:AddNode, PIPSC:RemoveNode, PIPSC:RetrievePKp, PIPSC: IsAddrOfController, PIPSC:AddContextAttr, PIPSC:GetContextAttr and pipsc:RevokeContext-Attr. $n_c$ manages the network with PIPSC.
### TABLE 3. Overview of the functions implemented by each SC, with sender, input arguments and result of the function when successfully executed.

| SC       | Function                  | Sender | Input Arguments                                                                 | Result                                      |
|----------|---------------------------|--------|---------------------------------------------------------------------------------|---------------------------------------------|
| PIPSC    | RequestAccess             | p_s    | Subject, Resource, ContextExp, Actions, Attributes, Auth<tr>                  | Calls PIPSC:EvaluateRequest                 |
|          | SaveDecision              | PIPSC  | Decision, Subject, Resource, ContextExp, Actions, PK_s                        | Saves Decision in PIPSC associated to PK_s and requested target |
|          | RevokeAccess              | n_c, n_p | Subject, Resource, ContextExp, Actions, PK_s                                 | Revokes Decision in PIPSC associated to PK_s and requested target |
|          | VerifyDecision            | n_s    | Subject, Resource, ContextExp, Actions, PK_s                                  | Verifies if PK_s has a Decision Permit to the requested target |
|          | SaveRequestLogObligation  | n_s    | ContextExp, Subject, Resource, PK_s, agent, generation, timestamp, action, outcome | Saves request header attributes from off-chain request in PIPSC |
| PDPSC    | EvaluateRequest           | PDPSC  | Subject, Resource, ContextExp, Actions, Attributes, Auth<tr>                  | Calls PDPSC:RetrievePKR, PIPSC:IsAddressOController, PAPSC:CreatePolicy and PIPSC:GetContextAttr. Then evaluate the request and yields Decision calling PIPSC:SaveDecision |
| PIPSC    | AddNode                   | n_c    | Addr, PK_s                                                                      | Adds a new node to the network, saving Addr, in PIPSC |
|          | RemoveNode                | n_c    | Addr, PK_s                                                                      | Removes a node from the network, deleting Addr, from PIPSC |
|          | RetrievePKp               | PDPSC  | Addr, Addr_p                                                                    | Searches for the addresses and retrieves PK_p from PIPSC |
|          | IsAddressOController      | PDPSC  | Addr_s                                                                          | Verifies if the signer is the legit n_c for n_p by checking whether the Addr, is associated to the signer Addr_s in PIPSC |
|          | AddContextAttr            | n_c, n_p | ContextAttr, ContextExp, Resource, Subject, Addr_p                           | Adds ContextAttr in PIPSC |
|          | RevokeContextAttr         | n_c, n_p | ContextAttr, ContextExp, Resource, Subject, Addr_p                           | Revokes ContextAttr in PIPSC and calls PIPSC:RevokeAccess |
|          | GetContextAttr            | PDPSC  | ContextExp, Resource, Subject, Addr_p                                          | Searches and gets ContextAttr in PIPSC |
| PAPSC    | CreatePolicy              | n_c    | Policy, ContextExp, Actions, Subject, Resource                                 | Adds Policy in PAPSC mapped to ContextExp,Actions, Subject and Resource |
|          | ChangePolicy              | n_c    | Policy or ContextExp, Actions, Subject, Resource                              | Changes Policy of the map relations |
|          | RemovePolicy              | n_c    | Policy or ContextExp, Actions, Subject, Resource                              | Removes Policy from PAPSC |
|          | LoadPolicy                | PDPSC  | ContextExp, Actions, Subject, Resource                                         | Loads Policy from PAPSC |

AddNode and PIPSC:RemoveNode, PIPSC:IsAddressOfController verifies whether the signer Addr_signer of a token Auth_signer is the same n_c which added n_p to the network. PIPSC also stores the ContextAttr according to the ContextExp. The relational database used to store the ContextAttr may differ for each ContextExp supported in the system. The functions to add, get and revoke ContextAttr are: PIPSC:AddContextAttr, PIPSC:GetContextAttr, PIPSC:RevokeContextAttr. We present in Section IV examples of two different ContextExp that customise the PIPSC according to the use cases.

### D. SETUP AND MAINTENANCE

At the system setup, the blockchain initial network is created with at least three controller nodes as required in the voting consensus adopted in the SmartAccess. Next, the controller nodes establish the functionality for SmartAccess by running setup functions in the steps illustrated in Fig. 3. In step 1, one n_c deploys all the contracts: PEPSC, PDPSC, PIPSC and PAPSC. In step 2, any n_c can add other controller and storage nodes by running the PIPSC:AddNode. Then, in step 3, each n_c adds its processor nodes, which are the professionals that work for the corresponding organisation. Finally, in step 4, any n_c can create a Policy by running PAPSC:CreatePolicy.

**FIGURE 3.** Setup of the smart contracts, nodes and policies. One controller node deploys all the smart contracts, then any controller can add new nodes to the network and create the access control policies.
M. Tuler de Oliveira et al.: SmartAccess: ABAC System for Medical Records Based on Smart Contracts

FIGURE 4. Flow diagram for access control and data retrieval with SmartAccess. Processor node $n_p$ requests read access to some data. This triggers communication between the various SC to permit/deny access and save the Decision. After obtaining permission, $n_p$ sends a request message off-chain to process the resource from $n_s$. $n_s$ verifies if the access has been permitted, returns the resource to $n_p$, and saves log information about the data processing. See the text for detailed explanations about the steps.

Once in the SmartAccess network, all controller nodes participate in the consensus mechanism and maintenance of the network. $n_c$ runs PIPSC:AddNode and PIPSC:RemoveNode to manage the controller and storage nodes. With the same functions, each $n_c$ manages its processor nodes. Moreover, any $n_c$ can run PAPSC:CreatePolicy, PAPSC:ChangePolicy and PAPSC:RemovePolicy to manage the policies, which must be in agreement with all the $n_c$ in the network. The policy agreements among controllers are achieved through the consensus mechanism, in which the transactions to run policy management are validated and stored in the blockchain in an auditable manner. See more details about the functions in Table 3.

E. ACCESS CONTROL FLOW

Fig. 4 shows the interactions between the various components during the evaluation of an access request. The flow is divided into three main steps: authentication, authorisation and data processing.

The processor node’s authentication happens off-chain. In step 1.a, a user $n_p$ sends credentials to be authenticated by the organisation ($n_c$) where $n_p$ works. $n_c$ authenticates and generates a signed authentication token Auth$^c_p$ containing the attributes of user $n_p$. This token is returned to the user in step 1.b and used in all further communication with the network.

The authorisation step, which is the main focus here, happens in the blockchain. It starts in step 2.a, when a user $n_p$,
in possession of Auth_p, runs PEPSC:RequestAccess. Then, in step 2.b, PEPSC calls PDPSC:EvaluateRequest and waits for a decision. In step 2.c, PDPSC takes the sender’s Addr_p and retrieves the respective PK_p from PIPSC. In step 2.d, PDPSC recovers the address of the signer Addr_s of the token Auth_p. Then, in step 2.e, PDPSC verifies if the signer is the legit n_c for n_p by checking whether Addr_p is associated with the signer’s Addr_c in PIPSC. In step 2.f, PDPSC verifies if the Attributes are legit by performing a hash function of the Attributes and PK_p, and comparing the result with the signed hash from Auth_p.

In step 2.g, PEPSC loads the Policy that is mapped to the ContextExp, Actions, SubjectID and ResourceID criteria. In step 2.h, PDPSC gets the ContextAttr required to evaluate the Policy. In step 2.i, PDPSC evaluates if the user has the Attributes and ContextAttr that comply with that Policy. Finally, in step 2.j, PDPSC calls PEPSC:SaveDecision to save the Decision related to PK_p and the requested target in the PEPSC. In step 2.k, the n_p receives the transaction TX hash after the evaluate PDPSC:EvaluateRequestFinished execution.

The actual data processing occurs in step 3.a, when the user n_p requests to process the resource by sending a message to n_s. This message is encrypted with PKs and signed by n_s. In step 3.b, n_s decrypts the message and calls PEPSC:VerifyDecision to verify if PK_p has a permit or denial decision associated with the target and resource in the PEPSC. If permitted, in step 3.c, n_s returns the resource to n_p, if not, the access will be denied. A similar process is used if n_p requests to create, update or delete data. After the response from n_s to n_p, in step 3.d n_s runs the PEPSC:SaveRequestLogObligation to save in the blockchain all the relevant information from the off-chain data processing to enable transparent auditing.

Once the n_p has Decision regarding the specific target, this Decision can be revoked. The revocation can be automatic if we define an expiration time to Decision or if n_p lose an attribute that granted the decision in the first place. We will see the revocation flow in detail in the use cases for each ContextExp.

IV. USE CASES: ACCESS CONTROL TO ELECTRONIC MEDICAL RECORDS

This section adopts SmartAccess to implement access control to share medical records in two scenarios. In the first use case, data access occurs with explicit patient consent, and in the second use case, access is consented implicitly during acute care. For each use case, we define PIPSC to store the corresponding ContextAttr according to the ContextExp.

The blockchain network comprises healthcare organisations (n_t), healthcare professionals and patients (n_p), and a centralised cloud service provider (n_c) with the off-chain Electronic Medical Records (EMR) for both use cases. Fig.5 and Fig.6 present examples of the operations required to grant and revoke access rights through the management of context attributes - these operations are explained in detail below.

A. ACCESS WITH PATIENT CONSENT

This use case illustrates the specific case that the patient n_p goes to a medical appointment with a healthcare professional n_p3. Before the appointment starts, n_p goes to a medical appointment with a healthcare professional n_p3. Before the appointment starts, n_p0 gives consent to n_p3 to access their EMR. A patient can only give and revoke consent to their own resources. The consent can apply to the entire EMR or only to a part of the EMR by specifying the SubjectID or the ResourceID, respectively. We designed the ‘ContextExp’ as ‘consented session’, where the Policy has one rule that n_p0 must have patient consent as ContextAttr saved on PIPSC associated to the Addr_p.

Fig. 5 shows the interaction of the patient with PIPSC to give access consent to n_p3. After authentication, the patient runs PIPSC:AddContextAttr(Consent, ResourceID, Addr_{n_p3}) to save in PIPSC the ContextAttr=Consent associated to ResourceID and Addr_{n_p3}. Later, n_p3 request to access the the specific ResourceID following the access control flow described in Fig. 4. The access is permitted since the n_p3 has valid ContextAttr and the other Attributes match the Policy. Access can have an expiration time or be revoked by the patient as follows: n_p0 runs the PIPSC:RevokeContextAttr(Consent, ResourceID, Addr_{n_p3}), that removes the ContextAttr and then triggers the access permission revocation PEPSC:RevokeAccess (ResourceID, Addr_{n_p3}) to remove already-permit accesses. If the n_p3 tries to access the patient’s data after the ContextAttr is revoked the decision will be ‘Deny’.

B. ACCESS DURING ACUTE CARE

In this use case, the patient needs acute care, so access to the EMR is implicitly permitted for the patient’s sake (fig.6). Multiple professionals treat a patient during an emergency,
FIGURE 6. Example of ContextAttr flow in the access during acute care emergency use case. Professionals and organisations involved in acute care can start an emergency access session, add and revoke new professionals to the emergency and end the emergency access session according to the data needed for the treatment. The professionals have the decision ‘Permit’ when also the ContextAttr legitimates their participation in the patient emergency. If the ContextAttr is revoked, the professional has access to ‘Deny’.

requireing dynamic access control to the patient EMR across healthcare professionals from different organisations. The entire patient’s EMR under emergency is referenced by SubjectID.

For this use case, we have four main nodes involved in the emergency treatment: healthcare professionals from call centre np1, ambulance np2 and hospital np3, and hospital controller np3. The professionals perform the cross-organisation sharing of data according to the data needed during the treatment. As soon as the professionals finish their task in the patient treatment, the professionals should have no longer access to the data.

Then we created ContextExp=Emergency session, which covers the time window since the patient requests emergency treatment until discharge. In PIPS C the professionals are added and revoked to a list of np involved in the patient’s emergency session. The call centre and hospital professionals may add the ContextAttr=StartEmergency for a patient who has the EMR related to SubjectID. Thus, once becoming part of the emergency session, any np can ‘add’ and ‘revoke’ another np from the emergency session. When the emergency session is over, the hospital organisation uses the ContextAttr=EndEmergency to revoke all the accesses of the remaining np involved in their emergency.

Fig.6 shows the interactions of the nodes with PIPS C. To start the emergency session, np1 runs PIPS C: AddContextAttr(StartEmergency, SubjectID, Addrnp1). Then, np1 requests an ambulance and adds np2 to the emergency session, running PIPS C:AddContextAttr function again for the Addrnp2. When np2 starts treating the patient, np2 removes np1 from that emergency session by running PIPS C:RevokeContextAttr(EmergencyMember, SubjectID, Addrnp2) which also revokes the access permission that was once granted to Addrnp1. The same happens for the ambulance and hospital professionals:
n_p2 adds n_p3, then n_p3 revokes n_p2. Finally n_c runs PIPSC:RevokeContextAttr(EndEmergency, SubjectID, Addr_{p3}) to finalise the emergency session.

V. THREAT MODEL AND SECURITY ANALYSIS

We consider two types of threats in our system. The first type of threat is inherited from the blockchain, consisting of attacks on the blockchain consensus and ledger, blockchain peer-to-peer (P2P) network, SC and wallets [34]. Regarding consensus attacks, a mitigation approach is to use consensus mechanisms that guarantee Byzantine fault tolerance. The attacks regarding SC and wallets occur due to faulty code development and exploitable vulnerabilities, which is out of this paper’s scope. For a future integration of SmartAccess and an EMR application, the SC must be extensively verified against exploitable errors. The second type of threat consists of attempts to bypass access control through impersonation and man-in-the-middle (MITM) attacks [35]. Here we focus on the threats to access control, where a corrupted n_p node impersonates a user, uses fake attributes or bypasses the authorisation steps. We state the following proposition, and below, we analyse the soundness of SmartAccess considering three attacks: Impersonation Attack, Fake Attributes Attack and Reuse Permission Attack. We denote the adversary as ADV and assume that the signature scheme is EUF-CMA secure [36] and the public key cryptosystem is IND-CCA2 secure [37].

A. IMPERSONATION ATTACK SOUNDNESS

Proposition 1: Let ADV be a corrupted n_p node that listens to the connection between the n_p, denoted as B, and n_c, denoted as C. B sends its credentials and receives Auth_{B_C} from C. ADV steals B’s token and runs PEPSC:RequestAccess to access a resource. We assume that ADV has no access to PK_B, then ADV has a negligible probability of successfully launching an Impersonation Attack.

Analyses. ADV launches an Impersonation Attack by running the PEPSC:RequestAccess function with the inputs: ContextExp, Actions, ResourceID, Attributes and Auth_{C} = \sigma_C(Hash(PK_B, Attributes)), where \sigma_C denoted the signature of C of the public key and attributes from B. PEPSC calls PDPSC:EvaluateRequest, which retrieves the public key of the sender PK_Adv and compares Hash(PK_Adv, Attributes) with Hash(PK_B, Attributes) from the signed Auth_C. Given the security of the one-way random hash function, there is a negligible probability that the hashes result is the same value.

B. FAKE ATTRIBUTES ATTACK SOUNDNESS

Proposition 2: Let ADV be a corrupted node that reads the policies in PAPSC and forges its Attributes to match with a Policy. Assuming that the signature scheme in Auth_{ADV} is EUF-CMA secure, the ADV has a negligible probability of successfully launching a Fake Attributes Attack.

Analyses: ADV launches Fake Attributes Attack after been authenticated and receiving legit Auth_{ADV} = \sigma_{C}(Hash(PK_{ADV}, Attributes)). ADV runs PEPSC:RequestAccess using altered Attributes_{fake} as input. As in Proposition 1, the comparison of the Hash results between the forged Attributes_{fake} and the Hash from Auth_{ADV} has a negligible probability of matching. The ADV can then tamper the Auth_{ADV} using the forged Attributes_{fake} instead of the legitimate Attributes. However, under the assumption of the secure digital signature scheme, ADV fails to forge a valid signature \sigma_{C}, and therefore, ADV fails to launch the Fake Attributes Attack.

C. REUSE PERMIT DECISION ATTACK SOUNDNESS

Proposition 3: Let ADV be a corrupted node that sends a message RequestData off-chain to interact with n_i without passing through the authorisation step in the blockchain. We assume that ADV has no valid access Decision = Permit for the requested target in the PEPSC. Therefore, ADV fails to launch the Reuse Permit Decision Attack.

Analyses: ADV launches Reuse Permit Decision Attack by performing a request to the n_i consisting of a RequestData message encrypted with public key of storage PK_i and signed by the PK_{ADV}. The n_i decrypts the message and runs PEPSC:VerifyDecision to check if ADV has a valid permit Decision, for the requested target and associated with the PK_{ADV}. Even if there is valid permission, the input arguments in the RequestData message must be the same as the target associated with the valid Decision. Thus, ADV cannot reuse valid permit access given to a different Resource or perform other actions than were permitted. Moreover, the PEPSC:VerifyDecision function compares the timestamp of the message against the permission expiration time. Thus ADV cannot use an old and expired access permission either. Therefore, ADV cannot successfully perform a Reuse Permit Decision Attack.

VI. IMPLEMENTATION

We have implemented a prototype of SmartAccess in Solidity [38] language and deployed and tested the prototype in an Ethereum Virtual Machine (EVM)-based blockchain system. Below we describe the implementation of the SC of the SmartAccess, the representation of the XACML policies and attributes using the Solidity language and how the policy evaluation is done.

We have implemented all the SC and each component described in Table 3 and Fig. 2 using Solidity language. The implementation of PEPSC, PDPSC, PAPSC and PIPSC for the two use cases are available on GitHub [39].

A. POLICY REPRESENTATION

In the prototype implementation, the policy is represented in two formats: the Policy representation to evaluate the Attributes of the user and logical operations mapped as code to evaluate the ContextAttr in PIPSC.
First, we express the logical size-efficient policy representation using an array of 8 bits, Policy[i], where the index \( i = \{0, \ldots, 7\} \). The controller nodes define a mask that adds meaning for each bit. Each bit of the array represents a rule of the policy. If the bit is 1, \( n_p \) must have the attribute, if the bit is 0, \( n_p \) is not required to have the attribute. For example, \( i = 0 \) represents the attribute ‘Hospital professional’, then Policy[0]=1 the user is a hospital professional and Policy[0]=0 means the user is not.

To evaluate the policy, the \( n_p \) needs an Auth\( p \) to perform an access attempt, which contains an array of 8 bits as the logical representation of the Attributes. Logically, the same mask used for the Policy is used to represent the Attributes. During PDPSC:EvaluateRequest, the decision process executes an AND logical operation between the two arrays, Policy and Attributes. The AND operation between two arrays yields ‘1’, where both indexes are ‘1’. Thus, \( n_p \) needs to have the ‘1’ in the same position in the Attributes as in the Policy.

Fig. 7 shows an example of policy mask, where \( i = 0 \) is the rule Role \( \Rightarrow \) “Doctor” and \( i = 7 \) is the rule Work shift \( \Rightarrow \) “Active”. Moreover, it shows an example of evaluation of a policy using the array AND operator. Since the output is different from the Policy, the Decision is Deny.

Second, each ContextExp requires additional evaluation of ContextAttr on PIPSC, as shown in the Section IV. For the ContextExp logical operations are mapped as code in PIPSC. For that, PIPSC have additional storage to save, get and delete ContextAttr for each ContextExp.

VII. EVALUATION

Here we present an initial evaluation of SmartAccess using the prototype (Section VI). We started by presenting the setup of the blockchain networks, then we deployed the SC and ran experiments to assess the implementation regarding performance, latency, size and scalability.

A. EXPERIMENTAL SETUP

The experiments were conducted on a Macbook Pro with M1 processor and 16GB of RAM. We created three private blockchain networks for the experiments using.

![TABLE 7. Example of a policy mask and Policy AND Attributes evaluation. The output of the evaluation is different from the Policy. Therefore the Attributes do not comply with the Policy.](image)

**FIGURE 7.** Example of a policy mask and Policy AND Attributes evaluation. The output of the evaluation is different from the Policy. Therefore the Attributes do not comply with the Policy.
and become available on the blockchain. This experiment indicates how long it would take for a user to pass through the access control (PEPSC:RequestAccess and PEPSC:VerifyDecision). The transaction latency is calculated as the difference between the timestamp of the block in which the transaction was mined and the transaction’s timestamp.

The size experiments measure the amount of data generated by the transactions and the mined blocks. This experiment provides insight into what to expect regarding growth and storage usage from the blockchain system utilising SmartAccess.

Finally, the scalability experiments measure how the TPS behaves for an increasing number of rules, attributes and contextual attributes that need to be verified during the policy evaluation process. We ran experiments with 64, 128 and 256 attributes and 1, 10 and 100 contextual attributes.

For each experiment, we ran 11 rounds, sending 5000 transactions to each of the three blockchain networks. The first round was used as a warm-up, and the results of this round were excluded from the final analysis. In total, 55000 transactions were created to run the four functions of the smart contracts (PEPSC:RequestAccess, PEPSC:VerifyDecision, Baseline SET and Baseline GET).

An overview of average results is presented in Table 4. Fig. 8 shows TPS results for the performance experiment. TPS results were similar among the different networks, although RAFT consensus performed slightly better on average. This result is expected, since the RAFT consensus performs fewer verifications and is not Byzantine fault tolerant (BFT). The results show that PEPSC:RequestAccess has significantly lower TPS than the baseline contracts. This TPS decrease is also expected since PEPSC:RequestAccess performs several steps to yield a decision. Note that this is different for PEPSC:VerifyDecision, which only needs to verify the access Decision. Overall, the average TPS was 250 for PEPSC:RequestAccess and 290 for PEPSC:VerifyDecision. Furthermore, in table 4 we can see a high increase in the gas usage for the PEPSC:RequestAccess, which shows that it performs more computational procedures, while losing around 15% in TPS compared to the baseline functions.

The latency results are expressed in seconds in table 4. For IBFT and QBFT consensus, the latency is, on average, approximately 2s (1 second spent for each block required to mine the transactions). The results using RAFT consensus show that it takes approximately 1s (one block). There were no differences in latency for the different functions of the baseline SC and SmartAccess contracts.

Table 4 shows the average sizes of transactions and blocks for IBFT, QBFT and RAFT consensus. The transactions to execute PEPSC:RequestAccess generate more data than the transactions to run the baseline contract functions. Moreover, from the transactions and block sizes, the growth of the network using the SmartAccess contracts is expected to be bigger than the baseline, following a linear growth.

In the scalability experiments, we deployed only IBFT consensus because byzantine fault tolerant consensus are more secure and widely adopted in private blockchain systems.
TABLE 4. Experiment results regarding performance, latency and size for the baseline contracts and the SmartAccess functions with different blockchain deployments. Average for 10 runs with 5000 transactions each. Tx=transaction, TPS=transactions per second.

| Consensus | Function               | Performance          | Latency         | Size   |
|-----------|------------------------|----------------------|-----------------|--------|
|           |                         | Tx gas               | Block gas (x10^8) | TPS (s) | Time (s) | Tx (KB) | Block (KB) |
| IBFT      | Baseline GET           | 21937                | 6477            | 295    | 1.9984   | 0.944   | 41.05     |
|           | Baseline SET           | 26798                | 7907            | 295    | 1.9985   | 0.946   | 41.03     |
|           | PEPSC:RequestAccess    | 164122               | 38483           | 252    | 1.9831   | 6.139   | 122.73    |
|           | PEPSC:VerifyDecision   | 31765                | 9141            | 288    | 1.9981   | 1.531   | 112.83    |
| QBFT      | Baseline GET           | 21937                | 6370            | 295    | 1.9984   | 0.944   | 40.49     |
|           | Baseline SET           | 26798                | 7805            | 295    | 1.9981   | 0.946   | 40.70     |
|           | PEPSC:RequestAccess    | 164122               | 40083           | 249    | 1.9809   | 6.139   | 120.86    |
|           | PEPSC:VerifyDecision   | 31765                | 9003            | 285    | 1.9985   | 1.531   | 111.52    |
| RAFT      | Baseline GET           | 21937                | 6529            | 299    | 1.0202   | 0.944   | 41.20     |
|           | Baseline SET           | 26798                | 7975            | 300    | 1.0202   | 0.946   | 41.28     |
|           | PEPSC:RequestAccess    | 164122               | 44116           | 269    | 1.0410   | 6.139   | 130.29    |
|           | PEPSC:VerifyDecision   | 31765                | 9182            | 290    | 1.0220   | 1.531   | 113.35    |

Moreover, the results for IBFT and QBFT were similar. The results are presented in Figures 9 and 10. Fig. 9 shows the transactions per second (TPS) considering a verification of a Policy and Attributes with 64, 128 and 256 Attributes. The results show that, despite increasing the number of attributes, the TPS remains the same, hence providing scalability regarding the attributes supported by SmartAccess. Fig. 10 shows the TPS considering a policy that requires evaluation of 1, 10 and 100 contextual attributes. The results show that the TPS is inversely proportional to the number of ContextAttr. On average, the TPS for 1, 10 and 100 ContextAttr is 225, 213 and 101. For 100 ContextAttr, the TPS decreases more than 50%; however, it supports 100 times more attributes, handling more sophisticated access control policies.

C. COMPARISON WITH THE START-OF-THE-ART

For the experimental comparisons with state-of-the-art, we could not obtain any open-source code for the related work solutions presented in Section II-C. Nevertheless, we performed a comparison of performance, size and latency between our proposal and two access control functions that are Solidity-based and available as open-source. One is an implementation of the Role-based Access Control Smart Contract (RBACSC) [45], and the other is an implementation of the Access Control List Smart Contract (ACLSC) [46]. In the experiments, we only used the function PEPSC:RequestAccess, because it is analogous to the functions from RBACSC and ACLSC that evaluate a request for data access. Note, however, that these functions, in fact, are different in the three access control solutions. In particular, RBACSC and ACLSC evaluation is much simpler than SmartAccess, which supports complex ABAC policies. Table 4 describes the smart contract functions from RBACSC and ACLSC that we use to perform experiments in comparison with PEPSC:RequestAccess.

RBACSC defines a policy-neutral access-control mechanism based on roles and privileges, where the permissions to perform certain operations are assigned to specific roles. RBACSC implements multiple functions for access control per role. The RBACSC:OnlyAdminCanViewThis function performs access control before executing an action, similarly to PEPSC:RequestAccess. For example, the action could be to read a file. RBACSC:OnlyAdminCanViewThis checks if the sender (requester) has an adequate role before executing the action, in this case, the role of ‘administrator’. ACLSC implements a list of permissions associated with a system resource. The list specifies which users or system processes are granted access to which resources and what operations are allowed on a given resource. ACLSC implements various functions for access control using different lists of permissions. We chose the ACLSC:GetResourceMetadata function for comparison with PEPSC:RequestAccess. The chosen function returns the metadata of a given resource stored in the smart contract storage, and its execution is protected. When this function is called, the contract checks whether the sender (requester) belongs to the group of members with permission to execute ACLSC:GetResourceMetadata. The resource model is defined by another smart contract.

Conceptually, both RBACSC and ACLSC implementations only protect the execution of functions inside the smart contract, therefore for resources stored on-chain. This is similar to the PEPSC:RequestAccess function, which requests permission to process a resource and saves the decision in the smart contract PEPSC. The difference between our proposal and the other two functions is that SmartAccess protects resources stored off-chain, and only the access control is performed on-chain.

Similarly to the experiment’s Section VII-A, we have deployed the same Quorum network running the IBFT consensus to run the analogous PEPSC, RBACSC and ACLSC smart contract functions. For SmartAccess contracts, we used Solidity version 0.8.21, while RBACSC used version 0.4.23, and ACLSC used version 0.8.1. The network has the same number of nodes (5) and time between mining blocks (one second). We ran 11 rounds (1 round for warm-up), sending 5000 ACLSC:GetResourceMetadata and
to be transparent to every network node. We consider that
the transparency supported by smart contracts enhances trust
in the system and enables real-time auditing of the data
processing agreements. Every new policy, or modification
in the policies regarding a shared resource, is transparent and
is validated through consensus by all nodes.

The evaluation indicated that our access control runs in a
reasonable time window, even for the increasing complexity
of the policies due to a larger number of attributes to be eval-
uated. Considering the network setup with IBFT consensus,
the overall latency to access the data is 4 seconds. In the case
of using contextual attributes, it needs two extra seconds to
add the ContextAttr in PEPSC, increasing the overall access
control to around 6s.

We acknowledge that our experiment and evaluation have
the limitation of running in a notebook with limited resources.
We then ran baseline contracts to be the base of compar-
ison for our results. Despite the decrease in performance
compared to the baseline contracts, the average through-
put of 220 TPS is reasonable to handle an inflow of
access control requests. For example, for the acute care
use case in the Netherlands, it is expected that roughly
110 emergencies per day [47]. Even in the worst-case sce-
nario, where the 110 emergencies and access requests hap-
pended simultaneously and with 100 contextual attributes
to be verified by SmartAccess, our experimental setup
would handle all the 110 PEPSC:RequestAccess and 110
PEPSC:VerifyDecision transactions in a short latency
time. The growth of the blockchain for the 110 accesses
would be less than 1MB per day.

Although the functionalities of PEPSC:RequestAccess
being more complex than RBACSC:OnlyAdminCanDo-
This and ACLSC:GetResourceMetadata, the TPS of
PEPSC:RequestAccess was significantly higher.
We hypothesise that this difference might be explainable by
the differences in implementations between the contracts and
the version of the compiler. The newest compiler version used
by SmartAccess has better optimisations than the previous
ones. Nevertheless, the results for gas and sizes were consist-
tent with the different complexity of the functions.

Regarding the transparency inherent to the system, some
privacy concerns appear firstly because the resources may
contain personal data, and secondly, because the transactions
may also contain some personal data of the professionals
or patients. The off-chain resource containing personal data
must be encrypted, and the identifiers of the encrypted files
ResourceID are related to SubjectID. The EMR application
running on top of the SmartAccess system is responsible for
having the relationship between the patient identifier and the
SubjectID of the resources. Thus, when the np sends the
request transaction, it only contains the SubjectID and
ResourceID, keeping the patient pseudo-anonymous in the
blockchain. We designed SmartAccess to address all nodes
using the pseudo-anonymous addresses, which means that a
person is linked to a Addrnp and PKnp, but no actual name or
the real-world identity is known unless the n which added

VIII. DISCUSSION
Access control systems usually keep their policies secret,
arguing that secrecy helps protect the system, the so-called
‘security through obscurity’. Nowadays, the practice is
frowned upon by the necessity of transparency and auditabil-
ity to enhance trust in the system. SmartAccess is designed

FIGURE 11. Transactions Per Second (TPS) of the three access request
functions of PEPSC, RBACSC and ACLSC in 10 runs.

RBACSC:OnlyAdminCanViewThis transactions to the
network. Similarly to the previous experiment, we also mea-
sured the performance, latency and size to compare with the
PEPSC:RequestAccess from figure 8 with IBFT con-
sensus. See the Section VII-A for more details about the
experimental setup.

An overview of the average results is presented in 5.
Fig.11 shows the TPS results from the three access control
requests. We observe that PEPSC:RequestAccess had
a higher TPS median result and a more stable distribution
range of results in [232, 277] TPS, in comparison with
RBACSC:OnlyAdminCanDoThis and ACLSC:GetRe-
sourceMetadata, which presented a dispersion in the
results from [167, 267] and from [176, 274] TPS,
respectively.

Table 5 also shows the results for the gas, latency
and size of the transactions and mined blocks. The
gas used by PEPSC:RequestAccess is much higher
than that used by RBACSC:OnlyAdminCanDoThis and
ACLSC:GetResourceMetadata. This is expected since the
last two functions verify only a single value before yield-
ing the decision. PEPSC:RequestAccess, however, per-
forms extra steps such as verifying the authentication token
and evaluating more complex policies, including contextual
attribute verification. The size of the transactions and blocks
is also higher for PEPSC:RequestAccess because of the
complexity of the transaction, which requires more input
parameters and performs more complex operations. However,
the latency between the generation of the transaction and
the transaction being mined followed the previous results
and stayed around 2 seconds for every function because it is
bounded by the IBFT consensus mechanism.
the \( n_p \) to the network. It is known as pseudo-anonymous addresses because the \( n_c \) must know the processor’s real identity, as the organisation knows the identification of its employee. The same applies to the patients, who must be added to the network by a \( n_c \). We assume that a national organisation could represent the \( n_c \) and be responsible for the authentication process of the patients’ \( n_p \), keeping their real identifiers private from the blockchain. For future work, we will investigate using a federated identity system for blockchain, such as Self-Sovereign Identity from Hyperledger Indy [48]. The pseudo-anonymous address also means that a combination of attributes available in the requests could be used to de-identify the person behind the node when combined with other sources of personal data. We consider it as a drawback of our proposal. However, any other system that uses pseudo-anonymous identifiers is subject to de-identification and privacy compromise. Thus we believe that using pseudo-anonymous addresses is a fair trade-off when we accomplish a transparent and auditable system.

We also assume that the usage of cryptography to protect the resources in the storage is necessary. Others have proposed innovative mechanisms that combine modern cryptographic schemes to protect data confidentiality with double-layer encryption, one for the data and the other for the cryptography keys [49]. For example, in our previous work [50], we used Dynamic Symmetric Searchable Encryption (DSSE) to enable the search of encrypted data combined with Attribute-based Encryption (ABE) to protect the DSSE keys used to encrypt the data. To interact with the DSSE and ABE, the system first performs access control using a centralised ABAC system. SmartAccess could be used as the access control layer (ABAC) above the cryptographic layers (DSSE and ABE) instead of only protecting access to the data. In future work, we intend to extend SmartAccess to replace the centralised ABAC solution.

Some blockchain proposals to store data processing logs rely on the assumption that \( n_s \) will send transactions to the blockchain only to record logs in there [27]. We consider this assumption unrealistic because there is no motivation for \( n_s \) to send a transaction for every request-response out of the blockchain. Therefore, SmartAccess logs are only about the access control requests and decisions done by the system. The logs related to data processing are recorded and maintained by the \( n_s \). During SmartAccess design, we could have opted to save information regarding the decision inside the payload of events inside each transaction. However, this way, the \( n_s \) would read the decision through the transactions without interacting with the smart contracts, and the verification logs would not be recorded. Because of that, we decided not to save the decision as payload; instead, we added PEPSC:VerifyDecision function where \( n_s \) executes the function to read the save decision inside the PEPSC. This way, \( n_s \) cannot know the decision unless it executes the function and generates the verification logs accordingly.

Finally, our solution adds complexity compared with ABAC implemented as a centralised service. However, current centralised solutions lack transparency and trust when sharing medical data cross-organisations. Moreover, we defined SmartAccess as an access control system to protect an EMR system with modular characteristics, making the system more flexible to changes for specific cases. All the SmartAccess requests have a clear purpose of processing defined by ContextExp. All the logs are associated with the purpose, which enables auditability across organisations and GDPR compliance. The two use cases described in this paper illustrate what can be designed using the SC. In future works, we will investigate how the SmartAccess can fit more use cases outside the healthcare scenario. Moreover, we will compare SmartAccess with traditional centralised access control mechanisms.

**IX. CONCLUSION**

This paper proposes SmartAccess, an Attribute-Based Access Control System for cross-organisation medical records sharing. The solution enables cross-organisation joint agreement over access policies, dynamic access control, transparency, and auditability. SmartAccess leverages the Attribute-Based Access Control model to implement four smart contracts that mimic the granularity of the model. We presented SmartAccess applied in two healthcare use cases: access with patient consent and access with implicit consent during acute care. We analysed the threat model and performed a security analysis in the initial evaluation. Through a proof-of-concept implementation of the SC, we demonstrated the feasibility of our proposal by analysing the complexity and scalability of the functions of the contracts and; by measuring the latency and throughput of the transactions to execute the access control functions with different blockchain network consensus setups. The results display expected overheads when executing the SmartAccess functions from the SC. Finally, our
implementation’s source code [39] is open to the community, which can facilitate further research leading to the adoption of SmartAccess in future applications.

REFERENCES

[1] (2016). Gdpr Art. 6 Lawfulness Processing. Accessed: Nov. 10, 2021. [Online]. Available: https://gdpr-info.eu/art-6-gdpr/

[2] (2016). What is a Data Controller or a Data Processor?. Accessed: Nov. 10, 2021. [Online]. Available: https://ec.europa.eu/info/law/law-topic/data-protection/privacy-terms/gdpr-art-6/

[3] (2016). Joint Controllers. Accessed: Nov. 10, 2021. [Online]. Available: https://gdpr-info.eu/what-is-data-processing-agreement/

[4] (2016). What is a Gdpr Data Processing Agreement?. Accessed: Nov. 16, 2021. [Online]. Available: https://gdpr.info/what-is-data-processing-agreement/

[5] A. Ullah, H. Dagdeviren, R. S. Cruz, and R. Deters, “Role-Based Access Control Smart Contract (Rbac-Sl)”.

[6] Y. Verginadis, A. Patiniotakis, Y. Verginadis, A. Michalas, P. Gouvas, G. Mentzas, and G. Schiefer, “Context-aware privacy-enforcement policy for PaaS-enabled access control,” IEEE Trans. Cloud Comput., vol. 10, no. 1, pp. 276–291, Jan. 2022.

[7] (2015). IPFS Powers Distributed Web. Accessed: Nov. 17, 2021. [Online]. Available: https://ipfs.io

[8] E. Psarra, I. Patiniotakis, Y. Verginadis, A. Michalas, P. Gouvas, G. Mentzas, and G. Schiefer, “Context-aware privacy-enforcement policy for PaaS-enabled access control,” IEEE Trans. Cloud Comput., vol. 10, no. 1, pp. 276–291, Jan. 2022.

[9] (2016). Gdpr Art. 6 Lawfulness Processing. Accessed: Nov. 10, 2021. [Online]. Available: https://gdpr-info.eu/art-6-gdpr/

[10] (2016). What is a Data Controller or a Data Processor?. Accessed: Nov. 10, 2021. [Online]. Available: https://ec.europa.eu/info/law/law-topic/data-protection/privacy-terms/gdpr-art-6/

[11] (2016). Joint Controllers. Accessed: Nov. 10, 2021. [Online]. Available: https://gdpr-info.eu/what-is-data-processing-agreement/

[12] A. Ghorbel, M. Ghorbel, and M. Jmaiel, “Accountable privacy preserving attribute-based access control for cloud services enforcing access control,” Int. J. Inf. Secur., vol. 21, pp. 1–20, Sep. 2021.

[13] P. Zhang, J. White, D. C. Schmidt, G. Lenz, and S. T. Rosenbloom, “IPFS: A resilient, fast, and decentralised file system,” J. Med. Internet Res., vol. 22, no. 8, Art. no. e13598.

[14] S. Rouhani, R. Belchior, R. S. Cruz, and R. Deters, “Distributed attribute-based control access system using permissioned blockchain,” World Wide Web, vol. 24, no. 5, pp. 1617–1644, Sep. 2021.

[15] A. Ghorbel, M. Ghorbel, and M. Jmaiel, “Accountable privacy preserving attribute-based access control for cloud services enforcing access control,” Int. J. Inf. Secur., vol. 21, pp. 1–20, Sep. 2021.

[16] P. Zhang, J. White, D. C. Schmidt, G. Lenz, and S. T. Rosenbloom, “IPFS: A resilient, fast, and decentralised file system,” J. Med. Internet Res., vol. 22, no. 8, Art. no. e13598.

[17] S. Rouhani, R. Belchior, R. S. Cruz, and R. Deters, “Distributed attribute-based access control system using permissioned blockchain,” World Wide Web, vol. 24, no. 5, pp. 1617–1644, Sep. 2021.

[18] P. Zhang, J. White, D. C. Schmidt, G. Lenz, and S. T. Rosenbloom, “IPFS: A resilient, fast, and decentralised file system,” J. Med. Internet Res., vol. 22, no. 8, Art. no. e13598.

[19] S. Rouhani, R. Belchior, R. S. Cruz, and R. Deters, “Distributed attribute-based access control system using permissioned blockchain,” World Wide Web, vol. 24, no. 5, pp. 1617–1644, Sep. 2021.

[20] A. Ghorbel, M. Ghorbel, and M. Jmaiel, “Accountable privacy preserving attribute-based access control for cloud services enforcing access control,” Int. J. Inf. Secur., vol. 21, pp. 1–20, Sep. 2021.

[21] P. Zhang, J. White, D. C. Schmidt, G. Lenz, and S. T. Rosenbloom, “IPFS: A resilient, fast, and decentralised file system,” J. Med. Internet Res., vol. 22, no. 8, Art. no. e13598.
MARCELA TULER DE OLIVEIRA graduated in telecommunications engineering from the Universidade Federal Fluminense, Rio de Janeiro, Brazil, in 2017. She received the M.S. degree in electrical and telecommunications engineering from the Graduate Program in Electrical and Telecommunications Engineering, Universidade Federal Fluminense. She is currently pursuing the Ph.D. degree with Amsterdam UMC Location AMC. She has participated in a student exchange program at the University of Florida, FL, USA. She has participates in the ASCLEPIOS H2020 Project (Trusted digital solutions and Cybersecurity in Health and Care) at Amsterdam UMC Location AMC. In her Ph.D. degree, she researches the usage of data encryption for securing and sharing healthcare and medical research data. Her main research interests include new-generation networks, sensor networks, network security, blockchain, and computational intelligence.

LÚCIO HENRIK AMORIM REIS is currently pursuing the degree in computer science with the Universidade Federal Fluminense (UFF), Niterói, Brazil. He is a Research Assistant at Amsterdam UMC Location AMC. His research interests include the applications of security mechanisms in healthcare, specially utilizing the blockchain technology, blockchain, and security of cloud-based systems.

YIANNIS VERGINADIS is currently an Assistant Professor in management information systems with the Department of Business Administration, Athens University of Economics and Business (AUEB), Athens, Greece. He has more than 15 years of experience in ICT projects in several research areas such as management of information systems, workflow management, complex event processing, cloud computing, and cloud security. He has a long-term cooperation, as a Senior Researcher with the Institute of Communication and Computer Systems (ICCS), Athens. His main research interests include management information systems, cloud, fog and osmotic computing, cloud security, smart contracts, blockchain, digital enterprise management, and knowledge management.

DIOGO MENEZES FERRAZANI MATOS (Member, IEEE) received the Ph.D. degree in electrical engineering from the Universidade Federal do Rio de Janeiro (COPPE/UFRJ), in 2017. From 2015 to 2016, he was a Sandwich Doctoral Fellow at the Laboratoire d’Informatique de Paris 6 (LIP6), Université Sorbonne, Campus Pierre et Marie Curie, Paris, France. He is currently a Professor at the Department of Telecommunications Engineering, Universidade Federal Fluminense (UFF), Niterói, Brazil. He received research grants from the Brazilian National Council for Scientific and Technological Development (CNPq), the Rio de Janeiro State’s Research Support Foundation (FAPERJ), and the Municipality of Niterói. His research interests include network security, next-generation networks, and the future Internet applications.

SÍLVIA DELGADO OLABARRIAGA is currently an Assistant Professor with the Department of Epidemiology and Data Science, Amsterdam University Medical Centers, University of Amsterdam, The Netherlands. She leads the e-science research line, supervises Ph.D. students, and teaches courses various education programs. Her research interest includes the usage of advanced information technology in scientific biomedical research.

---

**REFERENCES**

[47] Hartschting. Numbers and Facts About Vasculary Diseases. Accessed: Jun. 1, 2022. [Online]. Available: https://www.hartschting.nl/hart-en-vaatziekten/feiten-en-cijfers-hart-en-vaatziekten

[48] Hyperledger Indy. Accessed: Jun. 1, 2022. [Online]. Available: https://www.hyperledger.org/use/hyperledger-indy

[49] A. Michalas, “The lord of the shares: Combining attribute-based encryption and searchable encryption for flexible data sharing,” in Proc 34th ACM/SIGAPP Symp. Appl. Comput., Apr. 2019, pp. 146–155, doi: 10.1145/3297280.3297297.

[50] L. H. A. Reis, M. T. de Oliveira, D. M. F. Mattos, and S. D. Olabarriaga, “Private data sharing in a secure cloud-based application for acute stroke care,” in Proc. IEEE 34th Int. Symp. Computer-Based Med. Syst. (CBMS), Jun. 2021, pp. 568–573, doi: 10.1109/CBMS52027.2021.00039.