Article

Janus: Hierarchical Multi-blockchain Based Access Control (HMBAC) for Multi-Authority & Multi-Domain Environments

Vangelis Malamas [1,*], George Palaiologos [2], Panayiotis Kotzanikolaou [3], Mike Burmester[4] and Dimitrios Glynos[5]

[1] Department of Informatics, University of Piraeus, Greece; bagmalamas@unipi.gr
[2] CENSUS S.A.; gpalaiologos@census-labs.com
[3] Department of Informatics, University of Piraeus, Greece; pkotzani@unipi.gr
[4] Computer Science Department, Florida State University, USA; burmeste@cs.fsu.edu
[5] CENSUS S.A.; dimitris@census-labs.com
[*] Correspondence: bagmalamas@unipi.gr

Abstract: Although there are several access control systems in the literature for flexible policy management in multi-authority and multi-domain environments, achieving interoperability & scalability, without relying on strong trust assumptions, is still an open challenge. We present HMBAC, a distributed fine-grained access control model for shared and dynamic multi-authority and multi-domain environments, along with *Janus*, a practical system for HMBAC policy enforcement. The proposed HMBAC model supports: (a) dynamic trust management between different authorities; (b) flexible access control policy enforcement, defined at domain and cross-domain level; (c) a global source of truth for all entities, supported by an immutable, audit-friendly mechanism. Janus implements the HMBAC model and relies on the effective fusion of two core components. First, a *Hierarchical Multi-Blockchain* architecture that acts as a single access point that cannot be bypassed by users or authorities. Second, a *Multi-Authority Attribute Based Encryption* protocol that supports flexible shared multi-owner encryption, where attribute keys from different authorities are combined to decrypt data distributedly stored in different authorities. Our approach was implemented using Hyperledger Fabric as the underlying blockchain, with the system components placed in Kubernetes Docker container pods. We experimentally validated the effectiveness and efficiency of Janus, while fully reproducible artifacts of both our implementation and our measurements are provided.

Keywords: Access control; Blockchain; Multi-Blockchain; Multi-Authority; Multi-Domain; Attribute-Based Encryption

1. Introduction

In recent years, the continuous integration of new communication and computing technologies, along with the rapid adoption of new IoT devices, has led to an enormous increase in generated data. According to Statista [1], the amount of data recorded worldwide is expected to reach 181 ZB (zettabytes) in 2025, from 79 ZB in 2021. This huge amount of data is generated and processed by multiple authorities that belong to different, and sometimes critical, domains.

Complex and critical ecosystems, such as healthcare, civil aviation or the energy sector, must integrate the operational, functional, security, and privacy requirements of many types of users and at the same time reconcile the conflicting interests of stakeholders. In addition, in many cases, there is a need for users belonging to one authority to access data maintained by other authorities, within the same or in a different domain. Access to the underlying data should be granted based on the access control policy of the corresponding resource owner and on other restrictions that may be imposed at an inter- or cross-domain level.

For example, in the health sector users may be doctors, patients, medical and administrative personnel of healthcare providers (e.g. hospitals) or technical operators and remote administrators for connected medical devices. In these ecosystems, *granular access* to data and *interoperability* of services, are two basic prerequisites that greatly impact the privacy and security aspects of these systems.

Traditionally, access control systems have been implemented within enterprises. Initial methods for domain-wise access control were based on centralized cloud solutions [2]. Such solutions require centralized data centers to store and handle various types of data, including user identities, cryptographic keys, and access rights, as well as a trusted cloud system administration, to control the users' access rights and authorization. Although these systems offer interoperability, fundamental challenges of data security and management persisted. Specifically, there are three major concerns with this design approach. First, an attack on the centralized data center might result in a single point of failure, leading to massive data compromise [3]. Second, strong trust assumptions persist for the cloud infrastructure [4], e.g., cloud administrators are trusted not to abuse their position to gain unauthorized access to resources or interfere with the access privileges of legitimate users. Finally, it is not possible to maintain a globally trusted and immutable access log to allow auditing for possible data access privacy violations.

Motivation. As a result of the preceding discussion, more emphasis is required on innovative, decentralized, interoperable, and adaptable trust management systems. The following are two motivating examples.

Example 1: Fine grained access to healthcare data. The medical sector includes a variety of domains, such as regulators, hospitals, manufacturers, and insurance providers. At the same time, privacy-sensitive, health-related data may be created by various medical IoT devices such as health monitoring devices (e.g. glucose level, blood pressure, or sleep monitoring systems), or treatment devices (e.g. medical infusion pumps). Users of one authority may require granular access to data maintained by multiple authorities (stakeholders) and domains. For example, while a doctor is on duty, she may require access to the full medical history data maintained in multiple hospital databases, for a patient under emergency treatment; or, an administrator of a medical device manufacturer may require access to the configuration data of connected medical devices installed in different hospitals. In addition, regulators may require that the data be accessible from a single entrance platform, in order to log all data access requests and monitor privacy violations. At the same time, new sectors or stakeholders can dynamically join or leave the system. Note that users who are simultaneously members of multiple authorities may require special access. For example, a doctor in a hospital may also be a researcher at a university. This doctor would also require access to (statistical) health data maintained in *all* hospitals, for research purposes.

Example 2: Fine-grained access to data in a multi-domain supply chain. Consider a distributed supply chain tracking system, collectively used by supply chain stakeholders for collecting, integrating and analyzing data from a variety of sources. The various stakeholders have different requirements for data access. For example, a container shipping company requires access to data on cargo weight and quantity, while a retail end-user requires access to data on product provenance, storage, and transportation conditions (especially for sensitive merchandise). These systems must be interoperable, but also provide granular access to data. In addition, authorities such as customs or other governmental agencies may also require a single point of access for all queries to data, for global access verifiability, i.e. it must be possible for any entity, either inside or outside the system, to verify all access attempts to the data (either successful or not).

Contribution. The main contribution of this paper is to design and implement a secure and efficient access control model specifically targeted to dynamic, multi-domain and multi-authority environments. In particular:

- We formally define *Hierarchical Multi Blockchain based Access Control* (HMBAC), a novel access control model for multi-domain and multi-authority environments. HMBAC supports: *dynamic trust management* between different authorities; granular and flexible *domain-level access control policy enforcement*; a global source of truth for all entities, allowing for an *immutable and forensics-by-design auditing mechanism*.

- We provide *Janus* [5], an *artifact and reproducible* implementation of HMBAC for large-scale setup environments. Our implementation uses Hyperledger Fabric as the underlying blockchain technology. To support system orchestration, we developed APIs to allow *controlled user interaction* with the blockchain and *inter-blockchain synchronization*. User interaction with the API occurs through an Electron application, while global system orchestration is achieved through Docker containers and Kubernetes.
- Based on our implementation, we have conducted extensive efficiency analysis to actually verify the practicality and efficiency of the proposed access control system.
- Our system enforces *a single point of entry* that cannot be bypassed by users or authorities. This is achieved by modifying the well-known MA-ABE scheme of [6] in a *distributed two-step decryption procedure*. Part of the ABE decryption is performed by the multichain system itself by generating an attribute key linked to the requesting user on the fly. The user will be able to fully decrypt the data, provided that he/she has obtained the relevant attribute keys, which are required by the access policy. We developed a Go library as an add-on for Hashicorp Vault[1] and integrated it into Hyperledger Fabric. Secure attribute key storage is supported by embedding the keys in different Vault instances.

We note that although *Janus* utilizes the hierarchical multichain of [7] as a building block, the proposed HMBAC model is independent of the actual underlying multi-blockchain used; it is possible to implement HMBAC based on other underlying multi-blockchains.

2. Related work

During the last few years, several research attempts tried to provide fine-grained access control in the dynamic multi-authority and multi-domain setting, while maintaining interoperability [8]. Some of these works have focused on decentralization and privacy-preserving encryption [9], [10], [11]. Such targeted solutions enable adequate compatibility and fine-grained access control; however, they fail to combine credentials issued by independent authorities. Other solutions rely on privacy-preserving encryption, such as Attribute-Based Access Control (ABAC). For example, [12] proposes a decentralized MA-ABAC (DMA-ABAC) scheme for multi-domain healthcare ecosystems. Despite that authorities are able to control independently their security settings and enforce cross-domain policies, the lack of a mechanism obliging users to use the system for accessing the data, in combination with the absence of global verifiability, lead to strong trust assumptions. In [13] an ABAC solution is designed for the shared multi-owner setting, assuming a distributed setting with multiple authorities. The authorities own pieces of data and may issue attribute keys that users may combine to access data belonging to different authorities. Even though the proposed solution strengthens the restrictions for accessing data, it falls short on addressing the challenge of inter- and cross-domain policy enforcement, i.e., dynamically defining access policies both to control access for all authorities within a domain or for all authorities belonging in different domains. Many works try to solve this problem by using Multi-Authority Ciphertext Policy Attribute Based Encryption (MA-CP-ABE) schemes. For example, the authors of [7] present a hierarchical multichain access model suitable for multi-owner setting specifically for healthcare environments. With the use of MA-CP-ABE, in encrypting the data, the proposed model achieves both inter- and cross-domain policy management. However since multi-domain and multi-authority environments are inherently large-scale, the lack of an end-to-end implementation makes it hard to actually assess the efficiency and practicality of [7] in actual applications.

[1] https://www.vaultproject.io

Table 1. Comparative assessment of the various characteristics prevalent in the relevant literature.

Relevant literature	Access Model	Application environment	Multi-owner setting[1]	Dynamic Trust management[2]	Inter-Domain&Cross-Domain policy management	Enforces Single point of access[3]	Global source of truth[4]	Implementation (artifact)
[7]	Hierarchical Multichains	Healthcare	Yes (MA-CP-ABE)	Yes	Inter & Cross-Domain	Yes	Yes	No
[9]	Token-based	IoT	No	Yes	Inter-Domain	Yes	Yes	WAVE [14]
[10]	Token & Crypto-based	IoT	No	Yes	No (user-based)	No	Yes	Droplet [15]
[12]	DMA-ABAC	Healthcare	Yes (ABGS)	Yes	Cross-Domain	No	No	No
[13]	CP-ABKS	M-A	Yes (ABKS-SM)	Yes	No	No	No	No
[16]	Hierarchical	M-A	Yes (MA-CP-ABE)	No	Inter-Domain	No	No	No
[17]	Hybrid encryption	WBAN	Yes (MA-CP-ABE)	Yes	Inter-Domain	Yes	No	No
[18]	Attribute-based	Mobile Things	Yes (MA-CP-ABE)	Yes	Inter-Domain	Yes	No	No
[19]	Hierarchical	M-A	Yes (MD-AC)	Yes	No	No	No	No
[20]	Tree structure	Smart City	Yes (CP-ABE)	Yes	Inter & Cross-Domain	No	No	No
[21]	SEMAAC	IoT & Healthcare	Yes (CP-ABE)	Yes	No	Yes	No	No
[22]	Attribute-based	IoT	Yes (MA-CP-ABE)	Yes	No (user-based)	No	No	No
[23]	Token-based	M-A	Yes (MA-CP-ABE)	Yes	Inter-Domain	No	Yes	No
[24]	ECC	IoT & Healthcare	Yes (MA-CP-ABE)	No	Inter-Domain	Yes	No	No
[25]	Attribute-based	Supply Chain	Yes (MA-CP-ABE)	Yes	No (user-based)	Yes	Yes	No
[26]	Attribute-based	IIoT	Yes (CP-ABE)	Yes	No (user-based)	Yes	No	No
[27]	ABSC	M-A	Yes (MA-CP-ABSC)	No	No (user-based)	Yes	No	No
[28]	FADB	Transportation	Yes (CP-ABE)	No	No	Yes	Yes	No
[29]	Credential-based	IoMT	No	Yes	No (user-based)	Yes	Yes	No
[11]	EACMS	Healthcare	No	Yes	No (user-based)	No	Yes	No
Our approach	HMBAC	M-A & M-D	Yes (MA-CP-ABE)	Yes	Inter & Cross-Domain	Yes	Yes	Janus [5]

[1] Combines credentials issued by independent authorities. [2] Dynamic join and leave of authorities in an efficient way. [3] Data can be accessed only through the system.
[4] All data access attempts are immutably recorded and globally verified.

In [16] a MA-CP-ABE scheme is proposed that supports range policy, which however maintains the need for a trusted central authority. In [17], authors adopt the idea of hybrid encryption to reduce the computational overhead of data encryption, using a symmetric key to encrypt data and a MA-CP-ABE mechanism to encrypt the symmetric key. The authors in [18], also propose a MA-CP-ABE scheme combined with an outsourced decryption and zero knowledge proof for the Internet of mobile things. In [22], another MA-CP-ABE scheme is proposed suitable for IoT applications and devices with low computational capabilities. Decryption operations are outsourced to fog for efficiency reasons, but no policy management is supported. The authors of [23] introduce a token-based access control scheme which uses smart contract and blockchain to generate decryption keys according to verified user attributes. In [24] Elliptic Curve Cryptography (ECC) and MA-CP-ABE are combined to create an access control system that supports the setting of multiple authorities, but the addition or removal of authorities remains inefficient. In [25] the authors propose a privacy-preserving MA-CP-ABE scheme for blockchain-based applications in the supply chain. This model achieves fine-grained access control and versatile authorization and also protects user's private key from leakage even when some attributes authorities fail. However, in most of the above solutions, trust expectations about global transaction verifiability persist. Strong trust assumptions are required to ensure that all access transactions to the encrypted data are immutably logged and may be globally verified by all entities. In [21] the authors also suggest an access model based on CP-ABE for IoT in healthcare and even though they achieve the collaboration of independent authorities they do not solve the need for inter and cross domain policy management. In [27] an Attribute-based Signcryption (ABSC) scheme is proposed that relies on a central certificate authority to verify the attribute authorities and thus maintains strong trust assumptions.

Other works use hierarchical attribute-based access models, e.g. [19,20,30–32]. For example, [19] presents a hierarchical Multi-Dimensional Access Control (MD-AC) model for the authorization of multiple participants in the cloud. Also [20] relies on a trusted third party (TTP) and an attribute mapping center (MC) incorporating CP-ABE to provide granular access to users. Both works use the cloud for efficiency reasons assuming strong trust. In general, although hierarchical attribute-based access models are flexible and scalable, they are not suitable for multi-authority, multi-domain environments, where roles may not have a global and strict hierarchy.

The current state-of-the-art leverages blockchain technology to provide a variety of fully autonomous and hybrid solutions [33,34]. Although blockchain technology is more difficult to administer may introduce efficiency issues, it provides, by design, global verifiability of data access transactions. An immutable ledger can ensure the integrity of transactions and data while also enforcing trust among multiple untrustworthy parties [35]. For example in [28] the authors propose a fine-grained access control scheme for transportation ecosystems based on blockchain, that embraces the multi-owner setting with the use of CP-ABE. The system provides global verifiability and data integrity, but cannot support dynamic joins and leaves of nodes, or a dynamic policy management. Hybrid solutions move some of the services previously supported by cloud providers to the blockchain. While this approach resolves some of the issues (e.g. data integrity), others problems, such as strong trust assumptions for the cloud operator, still remain [36–38]. Although autonomous solutions are entirely based on blockchain, they are still limited by the level of efficiency that can be supported [11,26,28,29]. In addition, practical implementations for schemes such as the ones discussed above hardly exist in the literature.

In conclusion as depicted in Table 1, the proposed HMBAC model is capable of supporting fine-grained access control for any multi-authority and multi-domain (M-A&M-D) environment, and it also supports flexible inter- and cross-domain policy management and enforcement. At the same time, our HMBAC implementation, *Janus*, is one of the few fine-gained M-A&M-D access control models that are supported by an artifact implementation.

3. Background

This section briefly describes the two fundamental components of our model: the MA-CP-ABE scheme and the Hierarchical Multi-Blockchain architecture.

3.1. MA-CP-ABE

Multi-Authority Ciphertext-Policy Attribute Based Encryption (MA-CP-ABE) was initially proposed by [39] as an application of Attribute Based Encryption in which any party can become an authority with no global coordination requirements. However, the scheme required a trusted central authority to collect all master private keys from all AAs to compute the collective secret terms for system initialization. The MA-CP-ABE was later extended by [6] introducing fully decentralized CP-ABE systems for both composite-order and prime-order groups by utilizing the user's Global Identifier (GID) in the key to resist collusion attempts. Several other works have extended the characteristics of the scheme to allow fine-grained data access with attribute revocation for cloud data storage [40], improved efficiency [41] and storage space saving by using hierarchical attributes to compress redundant ciphertext information ([42], [43]).

With an MA-CP-ABE scheme, multiple authorities agree on a set of global parameters GP and, based on these parameters, each authority X generates a public/secret key pair PK_X, SK_X. Data m can then be encrypted based on a mutually agreed access policy $\mathbb{P}$ (in the form of a matrix), using the public keys of all the authorities, i.e.: $ct = Enc(m, \mathbb{P}, \mathrm{GP}, \{PK\})$. With MA-CP-ABE schemes, any party can become an authority, and there is no requirement for a global root authority. More importantly, users can combine attributes issued by different authorities, provided that each user has a unique global identity GID. Any authority X may issue to any user U, attribute keys for an attribute $attr$, using its private key, the global parameters and the user identifier:

$$K_{U,attr} = KeyGen(\mathrm{GID_U}, \mathrm{GP}, attr, SK_X).$$

Finally, users can combine their attribute keys, issued by multiple authorities, to decrypt an ABE-encrypted ciphertext ct, provided that their set of attributes satisfies an access rule within $\mathbb{P}$, i.e.: $m = Dec(ct, \mathrm{GP}, \{K_{U,attr}\})$. Although the scheme enables a combination of attributes issued by different authorities, it remains collision resistant, meaning that different users cannot combine their attributes, since each attribute key is assigned to a different GID.

An example of MA-CP-ABE is illustrated in Fig. 1. Several organizations belonging to different domains may agree on inter-domain or cross-domain access policies. A domain-wise policy for hospitals may be, for example, to allow access to patient health for doctors at any hospital, if the patient is under emergency treatment. A cross-domain policy for the hospital domain may be to allow access to anonymized data to researchers; or, access to configuration data of medical devices to authorized manufacturer admins. Users may be given attribute keys from different organizations (authorities) and combine them, since

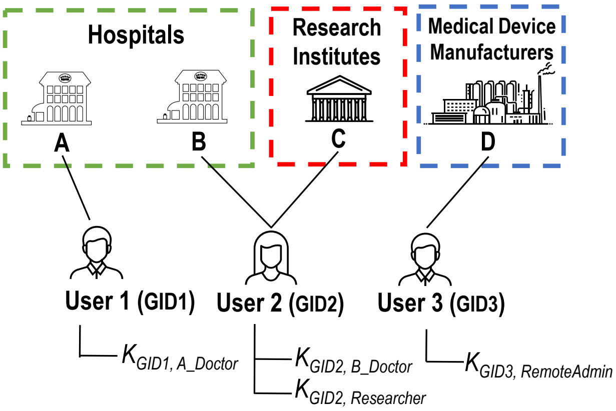

Figure 1. MA-ABE attribute binding and key generation

each attribute key is linked to the global identifier of a user. However, keys issued to different users cannot be combined. The challenge, from an implementation perspective, is to create an architecture that supports efficient MA-ABE decentralization and distribution of the decryption functionality, without compromising security.

In our system, we will utilize and properly modify the MA-CP-ABE scheme of [6] since: (i) It is a well recognized multi-authority ABE scheme which supports collision resistance, i.e. although the users may acquire attribute keys from different authorities, it is not possible to combine attribute keys that were issued to different users. (ii) It is supported by actual implementations. (iii) Its decryption process can be executed sequentially, where some attribute keys are applied first, for partial decryption. The partially decrypted data can then be fully decrypted in a second phase, possibly by another entity. In our system, we will exploit this to cryptographically enforce a single point of access for all users. Note that although *Janus* applies the MA-CP-ABE scheme of [6], other MA-ABE schemes which satisfy the above requirements could be applied to support additional functionality. For example in scenarios where data deletion is required, it is possible to extend Janus implementation by applying the MA-CP-ABE scheme of [44] which supports data deletion assurance.

3.2. Hierarchical Multi Blockchain

A Hierarchical Multi-Blockchain was proposed in [7] as a solution for fine-grained access to data in medical environments, with multiple participants (e.g. hospitals, medical device manufacturers, insurance companies, etc.), and multiple types of users (e.g. doctors, technical staff, etc.). This architecture was intended to enable autonomous administration of trusted medical data and transactions between mutually untrustworthy stakeholders, while at the same time providing a built-in forensic mechanism optimized for granular auditing. End users from a variety of health care domains can access and securely exchange medical data, provided that a domain-specific access policy is adhered to.

Fig. 2 illustrates the architecture of a Hierarchical Multi-Blockchain. At the first level, a Proxy Blockchain is the *single access point* for participating users. This provides interoperability between independently managed trust authorities and also acts as an immutable *single source of truth* for all transactions. At the second level, one or more Domain Blockchains enable each domain (e.g., hospitals, device manufacturers, insurance providers) to enforce their policies and provide fine-grained access control via Attribute Based Encryption (ABE). Databases are handled locally by each entity, and data are encrypted with the attribute keys based on the access policy of the corresponding domain. Services at all levels are implemented by smart contracts published on the relevant blockchain.

Other studies have also adapted hierarchical multi block-chain for secure data storage and sharing in various environments. For example, in [45] the authors propose such a model as a solution for massive data traffic in IoT environments, while in [46] and [47] a similar multi-domain blockchain architecture is presented as an answer to scalability

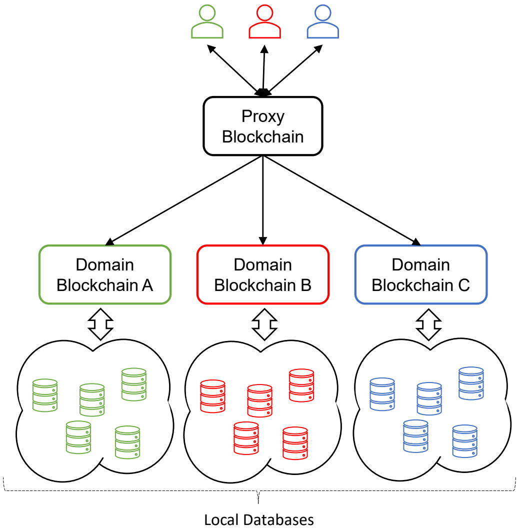

Figure 2. Hierarchical Multi Blockchain architecture [7]

issues. In [48], the authors propose an architecture to minimize IoT information loss by using multiple blockchain groupings, linking them in hierarchical chains. A similar scheme is used in [49] for fine-grained audit capability using sensor values to strengthen confidence. The authors in [50] use sidechains to ensure hierarchical fine-grained data access, but the proposed architecture lacks domain isolation and cannot be used for flexible, manageable domain-wise and cross-domain access policy enforcement. Other applications of hierarchical multi-blockchains involve domain level supervision systems [51].

Despite the growing evolution of Hierarchical Multi Block-chain applications, numerous open challenges remain, especially from the aspect of security (e.g. strong trust assumptions) and implementation (e.g., synchronization between multiple blockchains and inter-blockchain communication).

4. HMBAC Design & Implementation

First, we formally define the HMBAC model (Section 4.1) and, on the basis of it, we describe an HMBAC architecture design (Section 4.2), based on the underlying building blocks. The architecture is eventually translated into *Janus*, an actual HMBAC implementation (Section 4.3).

4.1. HMBAC access model

At a high level, the goal of HMBAC is to allow users belonging to different stakeholders (authorities) from different domains to have controlled access to data owned by multiple stakeholders. Moreover, the model must support interoperable use of credentials issued by different authorities, while the inter- and cross-domain policy management must be controlled at a domain level.

Hierarchical multiblockchains (Sect.3.2) play a central role in the HMBAC model. The Proxy Blockchain layer answers the first goal by allowing the interoperable use and verification of credentials issued and managed independently by different authorities and domains. Then, the various Domain Blockchains answer the second goal by allowing authorities belonging in different domains to define inter- and cross-domain policies for their data and to manage the authority membership in their domain, without affecting other domains.

Following the approach of [52], we define the HMBAC mo-del by representing the logical relations among the access control components, as shown in Fig. 3. A user (U) belonging to one or more authorities (Au) (for example, a hospital) is assigned attributes from a pool of attributes (UA). One or more authorities may issue attributes to users, which is depicted as the attribute authority (AA) relation in Fig. 3. Each authority is associated

doi:10.20944/preprints202212.0312.v1

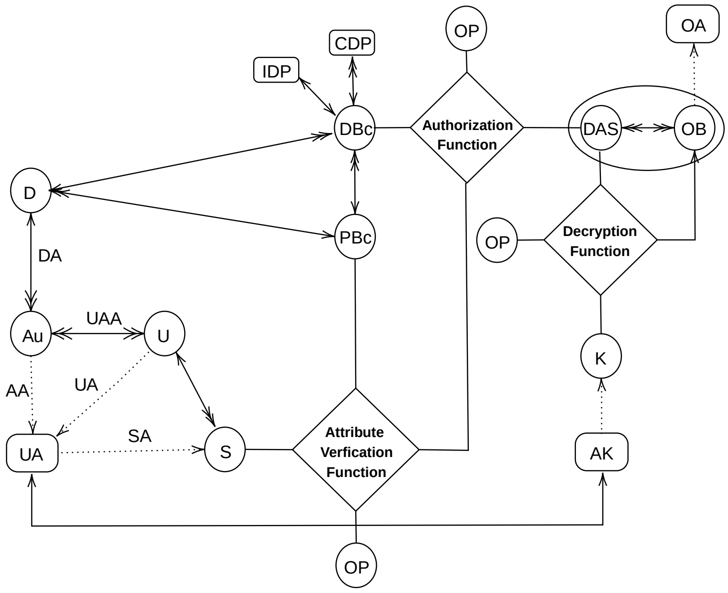

Figure 3. HMBAC access model: element sets and relations

with a single domain (D), and each domain contains multiple Au. A key element in the HMBAC model is the hierarchical multi-blockchain presented in Section 3.2. All domains, and thus all authorities, constitute the proxy blockchain (PBc), i.e. are the stakeholders for the first layer of the multi-blockchain. Then, at the second layer, various domain blockchains (DBc) may be constructed. Each group of authorities with similar characteristics form a different DBc (e.g one DBc is constructed by the hospital stakeholders, while another DBc is constructed by the manufacturers). Objects (OB), representing data or services accessible by users and operated by subjects (S), are encrypted with attribute keys (AK) created by the pool of UA. With the terms user attribute authorities (UAA) and subject attributes (SA) we represent the logical connection between users and subjects with the UA pool.

For a user to gain access to encrypted data, three checkpoints must be met. First, an *attribute verification function* (AVF) executed on the PBc will verify the validity of user attributes. Then the *authorization function* (AF), placed on the DBc, will verify whether the presented user's attributes are sufficient, based on the relevant inter-domain (IDP) and cross-domain (CDP) policies, to authorize the access request. Finally. the *decryption function* (DF) also executed in the relevant DBc of the data owner, will partially decrypt the data, which will be fully decrypted by the user, with the proper user attribute keys.

Table 2, summarizes the basic sets and functions of the proposed HMBAC model, as well as a formal analysis of the three functions specified for data access and decryption. Users, objects, and keys can be assigned attribute values directly from an attribute function *att* from a set of values in the range, denoted $Range(att_u)$, $Range(att_{OB})$, $Range(att_K)$ respectively. Users are assigned to multiple authorities (defined by many to many functions $direct_{UAu}$) and also authorities are assigned to one domain (defined by one to many functions $direct_{DAu}$).

Table 2. Basic sets and functions of HMBAC model

Basic Sets and Functions

-U, Au, S, K, D: finite sets of users, authorities, subjects, keys, domains
-UA, OA, AK: finite sets of user, object and keys attribute functions
- PBc, DBc: fine sets of Proxy and Domain blockchain services
-IDP, CDP: fine sets of inter and cross domain policies
-OB, OP, DAS: fine sets of objects, operations and data services
-$attType : UA = \{set\}$, defines user attributes to be set valued only.
-$attType : AK = \{set\}$, defines keys attributes to be set valued only

Each attribute att_U in UA maps users or authorities to a set of attribute values in $Range(att_U)$. Formally, $att_u : U \cup Au \to 2^{Range(att_U)}$

Each attribute att_{OB} in OA maps objects in OB to attributes values. Formally, $att_{OB} : OB \to 2^{Range(att_{OB})}$

Each attribute att_K in AK maps keys in K to attribute values. Formally, $att_K : K \to 2^{Range(att_K)}$

Direct $UAu : U \to 2^{Au}$, mapping each user to a set of authorities.
Direct $DAu : D \to 2^{Au}$, mapping each domain to a set of authorities.

Effective Attributes of Users, Subjects and Keys

For each attribute att_U in UA, $effectiveAu_{\ att_U} : Au \to 2^{Range(att_U)}$

For each attribute att_U in UA, $effectiveU_{\ att_U} : U \to 2^{Range(att_U)}$

$US : S \to U$, mapping each subject to a user

For each attribute att_U in UA, $effectiveS_{\ att_U} : S \to 2^{Range(att_U)}$, mapping each subject to a set of values for its $effectiveU_{\ att_U}$.

For each attribute att_K in AK, $effectiveK_{\ att_K} : K \to 2^{Range(att_K)}$.

Attribute verification function (AVF)

A subject $s \in S$ is allowed to perform $op \in OP_{AVF}$ on a service sr $\in PBc$ if the $effectiveS_{\ att_U} \in UA$. Formally, $OP_{AVF}(s : S,\ sr : PBc) = True$

Authorization function (AF)

A subject $s \in S$ is allowed to perform $op \in OP_{AF}$ on a service $sr \in DBc$ if the $effectiveS_{\ att_U}$ satisfy policies stated in $Auth_{DBc}(s : S,\ sr : IDP \cup CDP)$. Formally, $Auth_{DBc}(s : S,\ sr : IDP \cup CDP) = True$

Decryption function (DF)

A subject $s \in S$ is allowed to perform an operation $op \in OP_{DF}$ on an object $ob \in OB$ in data access services $ds \in DAS$, if $\{OP_{AVF}(s : S, sr : PBc) \cap Auth_{DBc}(s : S, sr : IDP \cup CDP)\} = True$ and has keys $k \in K$ such as $OP_{DF}(ob \in OB | ob_k \in K | s_k \in K) = True$.

4.2. HMBAC architecture design

The system architecture is designed with real-world requirements in mind, notably those of the medical sector. Note however that other digital environments can easily be supported.

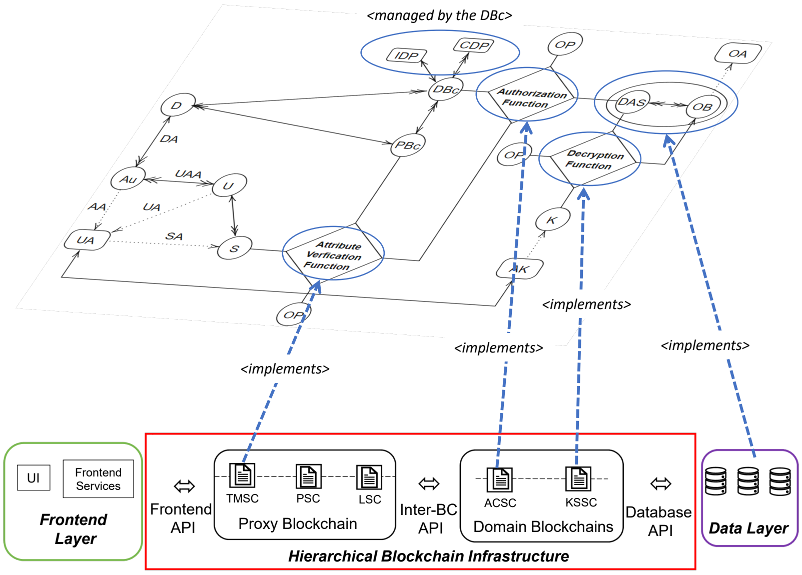

Figure 4. The proposed HMBAC architecture with the HMBAC model of Fig.3

In our implementation we consider two domains: *hospitals* and *medical device manufacturers*. Hospitals may involve users with various roles such as doctors, emergency doctors or researchers. Similarly, manufacturers may support various roles such as device technicians. According to the access control policies that may be defined within a domain, or cross-domain, granular access may be allowed. For example:

- *Access Rule 1: A doctor on duty may access all the medical records at all hospitals, of a patient under emergency treatment* (hospital-domain access rule).
- *Access Rule 2: A manufacturer's support technician may read or update the firmware of supported medical devices installed at any hospital* (cross-domain access rule).

Fig. 4 describes the proposed HMBAC architecture, as well as its mapping to the generic HMBAC model (presented in Fig.3). The architecture is comprised of three building blocks. The *Frontend Layer*, is a web application which allows authorized users to interact with the system and post data access queries. It consists of a web user interface (UI) and frontend services that support the communication between the frontend and the rest of the system.

The *Blockchain Infrastructure* is a middleware that implements all system services and provides controlled access to data, which are maintained off-chain individually by each stakeholder. It implements the hierarchical multi-blockchain, which consists of one Proxy Blockchain (PBC) and one or more Domain Blockchains (DBCs). As described in Section 3.2, the PBC acts as a single access point for users, while the DBCs enable the management, implementation and enforcement of flexible and granular access policies at the domain level. The integration of the blockchain components is supported by special-purpose APIs both for inter- blockchain synchronization (Inter-BC API), user interaction (Frontend API) and data (Database API).

Finally, the *Data Layer* contains the individually managed databases that store all data off-chain and ABE encrypted. As data are encrypted with ABE and the decryption process requires partial decryption through the corresponding DBC, the system enforces data access through the HMBAC system. Note that CA management at the stakeholder level is managed outside our system and our main goal is to offer credential interoperability, i.e. credentials issued from independent authorities are mutually trusted, without assuming a globally trusted root authority. In the following subsection, we describe in detail the services provided by each building block.

4.2.1. Frontend layer

A web UI enables users to login to the system and post data access queries. To access the system, two-factor authentication is enforced: the user must provide valid login credentials (e.g. a password) and a valid attribute certificate issued by a stakeholder. We assume that the user credentials are managed individually and stored securely by each user. The communication between the Frontend Layer and the Blockchain Infrastructure services is realized by endpoints implemented as Frontend Services and Frontend API. Finally, when the user eventually receives the partially decrypted response data via the Frontend API, the Frontend Services will grant access to the user of the attribute keys needed to fully decrypt the data.

4.2.2. Data layer

Stakeholders manage their data off-chain. Recall that data are MA-ABE encrypted based on predefined domain or cross-domain policies. To enforce accessing data only via HMBAC, for each domain a distinct *domain attribute key pair* is assigned. During the ABE data encryption, all policies are modified by applying an additional 'AND' rule with the corresponding domain attribute key. For example, data encrypted based on access Rule 1 defined in Section 4.2, for decryption would require the following attribute keys: K_{doctor}, K_{onDuty}, and $K_{hospitals}$. The key K_{doctor} corresponds to a long-term attribute and K_{onDuty} to a temporal attribute. Finally $K_{hospitals}$ is the hospital domain's attribute key, generated using the hospital domain's attribute key pair PK_H, SK_H.

4.2.3. Hierarchical Blockchain infrastructure

The *Proxy Blockchain* (PBC) receives the user requests via the Frontend API and implements three main services through smart contracts. User requests along with the provided attribute certificate(s) are handled by the *Proxy Smart Contract* (PSC). The PSC will first trigger a certificate validation process, executed by *Trust Management Smart Contract* (TMSC). The TMSC will validate the long-term (and possible temporal) attributes assigned to users via a typical challenge-response signature verification process. Note that users can access the HMBAC services only via an *authenticated channel* (the Frontend API) and after successful *attribute authentication* (by the PBC). If user attributes are verified, the PSC will then forward the request to the relevant Domain Blockchain for further processing and wait to receive the response via *Inter-BC API*. The response will eventually be sent back to the user via the Frontend API. The transaction history is recorded on the PBC. The *Logging Smart Contract* (LSC) creates a log for each incoming user request, until the transaction is completed.

The *Domain Blockchains* (DBC) may receive user requests only from the PBC via the Inter-BC API and implement the following services. Each DBC contains an *Access Control Smart Contract* (ACSC) which enforces the access control policy of the particular domain. This includes both intra-domain and cross-domain access policies that control access to data maintained by the domain's stakeholders. The ACSC checks if the user attributes (already validated at the PBC) are sufficient for the specific request, according to the predefined access policy. In this case, the request is forwarded to the relevant database(s) via *Database API*. Note that depending on the request, the API can retrieve ABE-encrypted data from multiple sources, for example, when a user is requesting data from multiple stakeholder's databases.

When the encrypted data return from the Data Layer, the Database API passes them to *Key Store Smart Contract* (KSSC), which has access to the relevant domain's attribute public/private key pair (e.g., PK_H, SK_H in the case of the Hospital DBC). The KSSC will first use the private key SK_H to generate the hospital domain attribute key ($K_{hospitals}$) based on the GID of the requesting user, to partially decrypt the data. The partially decrypted data will be forwarded to the PBC (via the Inter-BC API) and eventually to the user (via the Frontend API). The user must finally apply his attribute keys to fully decrypt and access the data (i.e., the attribute keys K_{doctor} and K_{onDuty} in our example).

4.3. Janus implementation

The implementation of the system relies on the integration of various technologies. For implementing the Frontend component, an *Electron*[2] application was developed, while blockchains (PBC and DBCs) were developed on the Hyperledger Fabric platform, with *Raft*[3] as the underlying consensus mechanism. Functionality is implemented through Smart Contracts developed in Javascript (the full open-source implementation can be found in [5] as a reproducible artifact).

Janus orchestration relies on Kubernetes[4]. For security and design modularity, all the components of the multi-blockchain infrastructure were developed in distinct Kubernetes Pods, thus providing software isolation and containerization. In particular, each smart contract, as well as the Frontend, the Inter-BC and the Database APIs are executed as separate Pods. To secure the interaction between Pods, an Ingress API supports TLS termination between the Pods.

To support the deployment of independent certificate infrastructures, each participating stakeholder establishes and maintains a *Certificate Authority* (CA), responsible for issuing and revoking certificates for their users who interact with the system. To simplify the deployment, one CA is considered to be a mutually trusted authority and is responsible for issuing certificates used by the system components (e.g. for Pods' TLS connections). Although this is not a strong requirement (e.g. it can be removed by applying cross-certification between the stakeholders), it simplifies the deployment process and it is a reasonable assumption for multi-domain environments. For example, in the healthcare sector, the ministry of health might have the role of the mutually trusted authority. In our system, we use *Hyperledger Fabric Certificate Authority* (provided by the Hyperledger platform) to implement the CAs of the stakeholders. Each CA runs as an instance for each entity in a separate Kubernetes Pod.

Hashicorp Vault[5] is used as an external application to issue, manage and store user and authority credentials and keys. Although it is an off-the-shelf solution, we designed an ABE plugin for Vault, written in Golang, to implement a two-step ABE decryption and support the partial decryption process via the KSSC. Additionally, we run different Vault instances, one for simulating user-side attribute key Vault storage and another for storing domain attribute keys, which is accessible only by the KSSC of each Domain blockchain.

Finally, the Frontend and the Inter-BC APIs use an instance of *RabbitMQ*[6] software, executed also in a separate Pod, for temporarily storing requests that remain pending in queues. The implementation design of Janus is depicted in Fig. 5 and described in detail in the following subsections.

4.3.1. Frontend application

This is a client-side Electron application, running locally by each user. The *User Interface* (UI) implements the presentation layer through a web-based interface developed in React.js. It enables users to login to the system using their appropriate credentials and, upon successful validation, to submit their requests.

In addition, the Electron app implements various support frontend services, developed in Node.js and running in the background at the client side. The *Blockchain-Related Mechanisms*, encompass all the functionality required for a user to communicate with the middleware. These mechanisms enable a user to create and sign a query and commit it for the purpose of generating a new transaction.

The *Authentication Mechanisms* supports user authentication to the system. To authenticate the credentials (attribute certificates) of a user, the Certificate Authority (CA) of the

[2] https://www.electronjs.org/
[3] https://raft.github.io/
[4] https://kubernetes.io/
[5] https://www.vaultproject.io/
[6] https://www.rabbitmq.com/

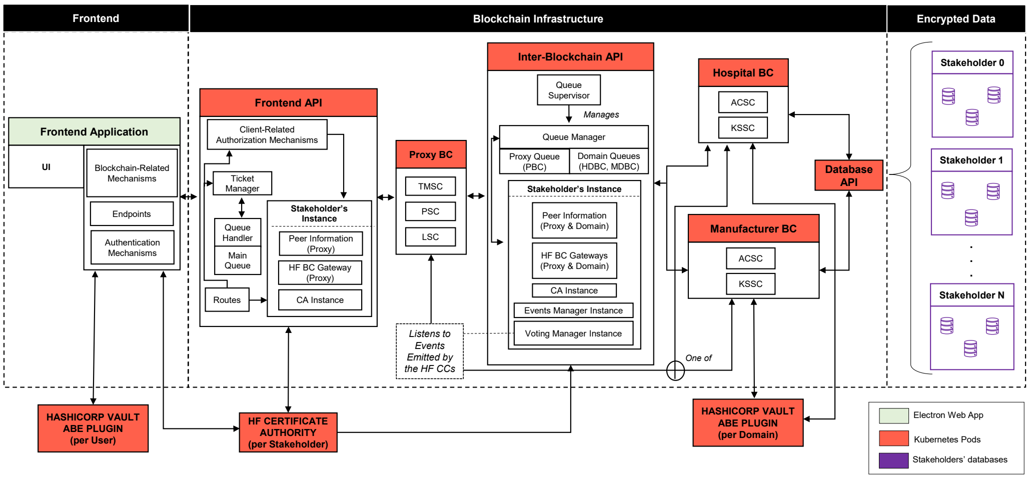

Figure 5. Janus implementation design based on the HMBAC architecture

relevant stakeholder must be involved. The frontend authentication mechanisms pass the credentials along with a signed challenge to the Frontend API, which communicates with the relevant CA in order to issue a token for the user and establish communication. The authentication mechanisms also support communication with the user's Vault instance, securely storing the user-side attribute keys.

Finally, the *Endpoints* represent the communication points between the UI and the Frontend API. Any traffic towards the API will be handled by Kubernetes and the Ingress web server.

4.3.2. Frontend API

This serves as a secure authenticated channel, allowing users to communicate with the PBC via the Frontend application. For each stakeholder, an instance is created including information about the node (*Peer Information*), the corresponding stakeholder's gateway (*HF BC Gateway*) and the instance of the CA (*HF Certificate Authority*) where the stakeholder's certificates are stored. Furthermore, *Client-related Authorization Mechanisms*, generate the authentication token issued by the CA to which the user belongs. This token is used to establish connections between the user and the PBC.

When a user submits a request, it is received by the *Routes* module and then queued in the *Main Queue* until served. The *Queue Handler* is responsible for data storage and retrieval to and from the Main Queue. When the Queue Handler authorizes the request to be forwarded, the request is retrieved from the Main Queue, and *Ticket Manager* generates a ticket for the user. Note that the Ticket Manager constantly checks for expired tickets.

The Frontend API is a RESTful API developed in Node.js using the Express framework. Since the Frontend API is running on a distinct Kubernetes Pod, its services are accessible to other Pods via the Kubernetes-exposed ports. In order to expose services on the web, an Ingress controller is used and the communication is TLS encrypted. For increased security, the TLS session is encrypted end-to-end between the Frontend application and the Frontend API. Thus, TLS is terminated on the Frontend Pod itself instead of the Ingress controller of the API.

4.3.3. Inter-Blockchain API

The Inter-Blockchain API is also developed in Node.js. Each stakeholder runs a different instance, executed on a distinct Kubernetes Pod, while its services are accessible to other Pods via the Kubernetes-exposed ports.

The main role of the Inter-Blockchain API is to enable the interaction between the Proxy and the Domain Blockchains. Handling and prioritizing requests between different blockchains is critical, as it can result in a significant performance decrease or even system

doi:10.20944/preprints202212.0312.v1

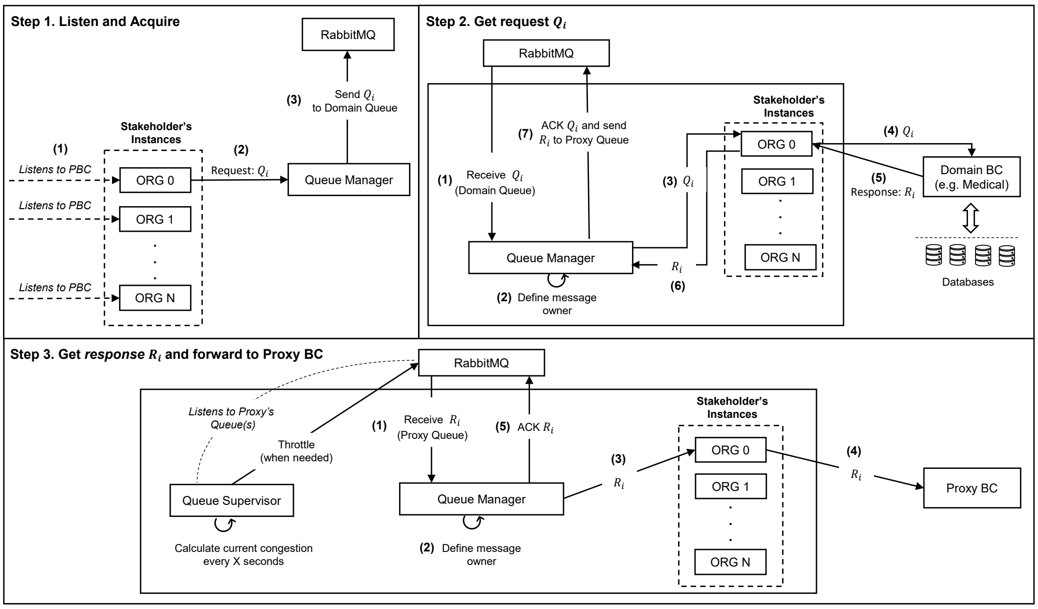

Figure 6. Inter Blockchain API Flow

failure. In addition, it supports stakeholders in making management decisions that may require agreement between stakeholders. Our solution addresses the aforementioned issues by integrating two critical processes into the Inter-Blockchain API: (a) Request Handling and Prioritization; and (b) Voting Management.

(a) Request Handling and Prioritization

For managing requests, the Inter-BC API, utilizes 2 general types of queue. The *Proxy Queue*, for every request that needs to be forwarded to the PBC and *Domain Queues*, for requests that need to be forwarded to one or more DBCs. Every DBC has its own Domain Queue, thus their number depends on the number of DBCs. The request handling flow is illustrated in Fig. 6 and is described below.

Step1: Listen and acquire. When an organization (*Stakeholder instance* in the Inter-Blockchain API) receives an event, it first verifies if it is the intended recipient. Next, it forwards the user query, say Q_i, to the *Queue Manager* in order to temporarily save the message and process it later when it is ready for consumption.

Step2: Get the request Q_i and forward it to the relevant DBC. When it is time for a request Q_i to be consumed by the appropriate DBC, the Queue Manager receives it (from the relevant Domain Queue of the Inter-BC API) and forwards it to the instance of the organization to which the requesting user belongs. The organization instance then forwards Q_i to the DBC of which the organization is a member. Upon completing the request, the response that was received from the DBC is then, again, sent to the Queue Manager in order to queue the new message and consume it, i.e. forward the response to the PBC when ready. The Queue Manager also sends an Acknowledgment message (ACK) to RabbitMQ to inform it that the message was successfully consumed. RabbitMQ receives the ACK and removes the message from the Domain Queue.

Step3: Get the response R_i and forward it to the PBC. When the response R_i of the query Q_i is received from the relevant DBC, it is ready to be consumed. The Queue Manager receives it through *Proxy Queue* and forwards it to the appropriate organization instance. Then, the organization receives the response and forwards it to the PBC to continue processing the response. When forwarding R_i back to the PBC, the Queue Manager sends an ACK to RabbitMQ to inform it that the message was successfully consumed. RabbitMQ receives the ACK and removes the message from the Proxy Queue. To maintain the flow healthy and avoid Denial-of-Service errors or attacks on the Proxy BC, a *Queue Supervisor* is utilized. The Queue Supervisor constantly monitors the Proxy Queue for spikes/congestion

doi:10.20944/preprints202212.0312.v1

in the network. As shown in Table 3, we define five types of congestion where each one is chosen based on the current level of congestion (CL).

Table 3. Congestion types

Congestion Types			
Scale	**Congestion Level (CL)**	**Monitor Interval**	**Throttle Multiplier**
Normal	$2 \leq CL$	5s	1
Low	$1 \leq CL < 2$	4s	0.7
Medium	$0.5 \leq CL < 1$	3s	0.4
High	$0.3 \leq CL < 0.5$	2s	0.1
Extreme	$CL < 0.3$	1s	0.01

Depending on the congestion type, the system adjusts, in order to handle the requests, avoiding DoS, and assuring that all requests will be served. The Congestion Level *CL* is calculated using the formula:

$$CL = \frac{max\ \#Concurrent\ Requests}{\#Queued\ Requests}$$

This means that, for example, if the PBC receives 1500 requests and accepts 400 concurrent requests, then 1100 of them will be pending in queue. The congestion level is then: $CL = 400/1100 = 0.36$, which is *High* according to Table 3. The system will lower the *monitor interval* from 5*sec* to 2*sec*, and throttle requests sent from IBC-API to PBC will be recalculated with the multiplier 0.1. If the default value is 400 then 40 concurrent requests will be sent until the PBC is dis-congested.

(b) Voting Management

To reach management decisions that require agreement between stakeholders, an asynchronous voting mechanism is implemented through smart contracts (described in Section 4.3.4 below). The Inter-Blockchain API enables the execution of this asynchronous model through the following components. As each stakeholder runs an instance of the Inter-Blockchain API, it deploys an instance of an *Event Manager* and a *Voting Manager*. The Event Manager continuously checks for new events related to active voting processes and expired elections. The Voting Manager fetches all the active elections related with the relevant stakeholder running the Inter-Blockchain instance, and it constantly awaits to receive relevant data (e.g. new votes). It also keeps a log of the submitted votes for each Election ID, and communicates with the PSC chaincode when majority is reached or time expiration occurs for a particular election.

4.3.4. Smart Contracts

As described in Section 4.2.3, the blockchain services are implemented through smart contracts. The PSC, TMSC, and LSC are stored at the Proxy Blockchain and shared among all stakeholders, while the ACSC and KSSC are stored at each Domain Blockchain and provide domain-specific functionality (see the relevant artifacts provided in [5] for details).

Proxy Smart Contract (PSC). This integrates the functions for *user validation*, *stakeholders' voting* and *request forwarding*. For user validation, the *validateUser*() function takes as input an array of *userCerts* provided by the user at login, and triggers the *getUserValidation*() function stored on the TMSC, in order to validate the provided certificate(s). Based on the outcome of the validation, the PSC returns the validated user's roles (short- and long-term). The voting mechanism is a crucial component of the system. Through this mechanism, stakeholders can reach management decisions including: a) adding/removing domains (and the relevant DBCs); b) adding/removing stakeholders to an existing DBC; c) modifying a (cross-) domain access policy; and d) giving access to logs for external auditors. Other processes that require broader consent can also be supported. For each stakeholder, the stakeholder administrator or a delegated external auditor may invoke the *majorityConsentInit*() function to propose a new election. It first checks if another election with the same payload is active and then calculates the ElectionID based on the provided payload. A new Election instance is created and added to the ledger, and also a ballot for each stakeholder. With *majorityClientVote*(), the administrator of each stakeholder votes in an Election by signing with the organizations' private key. The *updateElection()* function checks if the Election has been finalized based either on the predefined majority rules or the timeout set. The *requestAccess*() function handles two processes. First, it constructs the *requestDetails*, which is sent to the LSC for logging. Then it sends a *requestForward* event to the Inter-BC API to complete the request.

Trust Management Smart Contract (TMSC). This contains the functions that support trust management services. The *initLedger*() function is responsible for handling certificates and revocation lists of each stakeholder. It takes an *initPayload* argument and appends the corresponding data to the PBC. Additionally, it creates empty Access Control List (ACL) files for each organization. The *getUserValidation*() function takes as input either an ACL file (to verify temporal roles assigned to users) or a certificate (to verify long-term user roles). Note that users may have obtained certificates issued by different stakeholders, provided that all certificates of a user include the same unique global identifier (GID). In addition, it allows for the efficient revocation of user access through attribute certificate revocation lists issued by the relevant authorities, instead of applying costly attribute key revocation techniques. Finally, it communicates with the LSC in order to record the transaction on the blockchain. To allow for interoperability of credentials issued by different stakeholders, all root certificates of all stakeholders are stored in the PBC. For the addition or removal of CAs, the functions *addCA*() and *removeCA*() are triggered accordingly. Note that both functions require agreement between current stakeholders through the voting mechanism. Only after agreement has been achieved through the voting mechanism, the function *updateTrustAnchors*() will update the stakeholders' certificates in the PBC. The *majorityUpdate*(), invoked by the PSC, is called when an election ends (either by majority agreement or by timeout) for informing the TMSC.

Access Control Smart Contract (ACSC). Its main function is to enforce the predefined access policy, when users request access to data stored within the domain. The Inter-BC API forwards the request and triggers the *policyEnf*() function. Using the *data_ID* and the *roles* provided in the payload, it determines whether or not to grant access and forward the request to the KSSC.

Logging Smart Contract (LSC). This component enforces a *single source of truth* in our system. For each data access request, the *requestLog*() function is automatically triggered by the *requestAccess*() function of the PSC. Two main processes are supported by the LSC, *registration* and *retrieval* of logs. Log registration is utilized with the functions: *updateLog*(), for updating the details of uploaded stakeholders' Certificates and temporal ACLs; *updateRequestLog*() for updating existing request records; and *majorityUpdate*() for updating Election records. The *getUserRequestLog*() function implements log retrieval for users who want to access their request record. The *retrieveLogInit*() and *retrieveLogs*() functions are utilized for starting an access-granting Election, when an auditor requests access to the logs stored on PBC and the log retrieval accordingly.

Key Store Smart Contract (KSSC).

This component enforces the *single point of access* property in our system, in the following way. As described in Section 4.2.3, data are MA-CP-ABE encrypted, based on predefined access policies, which are modified to additionally require the decryption with the domain attribute key. The KSSC is the only component that may access the domain's attribute private key. The *requestData*() function is invoked by the ACSC to initiate the process, only after the user's roles have been verified and connects to the Database API to forward the *data_ID* of requested data.

4.3.5. Distributed ABE decryption implementation

As described in Section 4.2.2, our goal is to cryptographically enforce a single point of entry for the system users; which means that even if a user has all the roles required for accessing some data, an extra layer of encryption will prevent access to the data outside the Janus system. To achieve this, we modified the implementation of the MA-CP-ABE scheme of [6], by distributing the decryption functionality between the user and the domain blockchain. We used as a basis the Python implementation of the original scheme in the *Charm* encryption library.

Since directly applying ABE encryption and decryption is not efficient, we have applied a hybrid encryption approach, where the data are symmetrically encrypted, while the symmetric keys are ABE encrypted. At each stakeholder's database, each data item, say d_i, is initially encrypted with a distinct symmetric (AES) key k_i as: $c_i = AES(d_i, k_i)$. Then, each data encryption key k_i is ABE encrypted, based on all access policies that allow access to the particular item, which are extended to include the domain attribute key of the relevant domain. For example, assume that personal information d_i of a patient should be available to the patient's family doctor or any doctor in the case of emergency treatment of the patient. In that case, the key k_i would be ABE encrypted as follows:

$$e_1 = Enc\big(k_i, \mathbb{P}, \mathrm{GP}, \{PK_{doctor}, PK_{fDoctor}, PK_H\}\big)$$
$$e_2 = Enc\big(k_i, \mathbb{P}, \mathrm{GP}, \{PK_{doctor}, PK_{onDuty}, PK_H\}\big)$$

Each symmetrically encrypted data item c_i is sent to the DBC through the Database API, along with all ABE encryptions of k_i, in this example e_1, e_2. The KSSC has access to the domain's vault, where the hospital domain's attribute key pair PK_H, SK_H is stored. Using SK_H it will generate *on-the-fly*, the hospital domain attribute key for the requesting user, i.e.: $K_{U,hospitals} = KeyGen(\mathrm{GID_U}, \mathrm{GP}, attr{:}hospitals, SK_H)$ and use it to partially decrypt the data. The user will be able to actually decrypt the data, only if: (i) the KSSC has partially decrypted the data with the domain attribute key and (ii) the user has the relevant attribute keys for (at least) one of the above access policies, i.e. $\{K_{U,doctor}, K_{U,fDoctor}\}$ or $\{K_{U,doctor}, K_{U,onDuty}\}$.

We implemented the MA-ABE decryption scheme of [6] in Go as a Hashicorp Vault plug-in, and we integrated this into KSSC. The KSSC may trigger *sysDecrypt*(), executed in the domain's Vault instance, which generates the domain attribute key for a given GID and uses it to perform partial ABE decryption. In this way, the domain attribute key is accessible only for those requests that have already been authorized by ACSC. At the same time, it is never given to the users, to prevent off-system data access.

5. Efficiency analysis

Since HMBAC is targeted to fine-grained access for multi-auhoritiy, multi-domain environments, a practical implementation must be scalable to the number of authorities and domains. First, we analyze the scalability of the system in terms of system management. Then we benchmark the performance of Janus for different configurations and access request rates. All measurements can be reproduced through the Janus github repository [7].

5.1. System Scalability and Management

The modular design of the HMBAC architecture allows for scalable and efficient system management. Adding or removing users in Janus is handled independently by each organization (stakeholder). Each organization is able to issue attribute certificates and give access to the corresponding attribute keys to allow its users to: (i) post queries that will be accepted by the ACSC, based on the user's roles; and (ii) fully decrypt a response

[7] Benchmarks are fully reproducable via an automated script – see the 'System Benchmark' section of the 'readme' document on Janus repository [5]

that has been partially decrypted by the KSSC. Adding/removing stakeholders within an organization, or changing the access policy of the domain, is handled at the domain level. Due to the use of independent DBCs per domain, managing functions within a domain will not cascade to affect the other domains. The use of the voting mechanism enables setting up elections at a domain level and in addition to define a majority threshold at domain level for decisions affecting a particular DBC. Finally, adding new DBCs will require a majority voting by all the stakeholders and will affect all the domains, as this will require updating the smart contracts in the PBC.

5.2. Benchmarks

We conducted our evaluation on two different hardware configurations with varying resources, using the Linode cloud infrastructure. As depicted in Table 4, in the first H/W setup (S1), an AMD EPYC 7501 32-core processor @2GHz with 64 GB RAM is used. The second H/W setup (S2) is an environment with higher resources, based on an AMD EPYC 7702 64-core processor running at @2GHz with 512 GB RAM. As our implementation Janus utilizes eight (8) Kubernetes pods, where each Pod corresponds to an independently managed server, setup S1 (resp. S2) corresponds to 4 cores/8GB RAM (resp. 8 cores/64GB RAM) per server.

Table 4. H/W specs for testing

	CPU (# cores)		**RAM (GB)**	
	Total	Per Pod	Total	Per Pod
Setup S1	32	4	64	8
Setup S2	64	8	512	64

Both sets of configurations run Ubuntu 20.04.1 LTS OS and Kubernetes 1.20.11 was used for container orchestration. The multi block-chain components were developed in Hyperledger Fabric 2.4 beta with Raft as the underlying consensus algorithm and also fabric-ca-client 2.2.6, fabric-network 2.2.9 and fabric-gateway 0.1.0 were used for establishing communications.

Following the two access rule examples mentioned in Section 4, we created both inter-domain queries (e.g. *"Retrieve the medical record for patient P from all hospital databases"* and cross-domain queries (e.g. *"Update the firmware for medical device D of manufacturer M at all hospitals"*). Each database was running on a separate Pod and data were ABE encrypted. The initial ABE decryption was performed by the KSSC running on the relevant domain BC of the requesting user, as described in Section 4.3.5.

We measured the average end-to-end query response time for various sizes of queries, ranging from 2 up to 300 *concurrent* queries (req/sec), with an approximately even portion of inter-domain and cross-domain queries. Table 5 shows the average execution time for all scenarios tested. In addition, the table presents the time needed for the main subprocesses of the query-response process.

Table 5. Detailed performance evaluation for various scenarios of concurrent requests and h/w setups (*time in sec*)

# of concur. requests	2		10		20		40		60		80		100		200		300	
H/W setups	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2
Ticketing	0.004	0.002	0.003	0.002	0.002	0.002	0.002	0.003	0.003	0.005	0.025	0.008	0.018	0.015	0.015	0.024	0.032	0.02
Endorse	0.08	0.07	0.25	0.12	0.24	0.17	0.16	0.23	0.22	0.15	0.48	0.55	0.44	0.18	0.55	0.3	0.82	1.44
Commit	0.006	0.007	0.006	0.004	0.005	0.005	0.006	0.005	0.005	0.005	0.008	0.008	0.008	0.025	0.010	0.010	0.014	0.020
BC_RTT	2.19	2.2	2.41	2.22	2.47	2.07	2.95	2.49	3.19	2.76	3.33	2.74	4.22	3.76	6	4.63	8.09	5.62
Average	2.27	2.27	2.67	2.35	2.72	2.25	3.12	2.73	3.43	2.92	3.85	3.32	4.69	3.98	6.58	4.97	8.96	7.12
Min	2.26	2.27	2.58	2.28	1.88	2.1	2.02	2.92	1.62	1.61	1.88	1.98	2.04	2.45	1.74	1.54	1.65	2.33
Max	2.29	2.27	2.72	2.39	2.91	2.58	3.88	4.05	4.86	3.66	5.14	5.31	5.71	5.22	9.2	7.84	12.77	10.39

Ticketing, refers to the time required by the system to issue a ticket for a user. *Endorse*, is the time it takes for peers to receive a request and sign the result. *Commit*, is the time required by the orderer nodes to create a new block. Finally, *BC_RTT* is the time needed to execute all the required BC functions (smart contracts) and inter-BC communication. In addition, the minimum and maximum time required for a query is presented in each scenario, to exhibit the deviation from the average time. As expected, the most resource-intensive process is BC_RTT, which encompasses all subsystems, from Proxy BC up to the retrieval of encrypted data from the independently managed databases, as well as the partial decryption process using the domain keys.

However, the overall time increase is linear (see Fig. 7), which indicates the scalability of the HMBAC design.

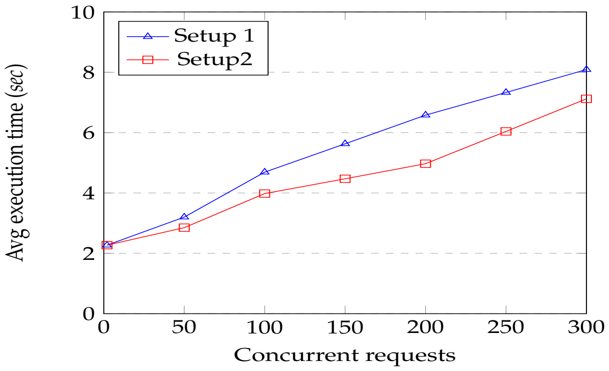

Figure 7. System Efficiency

Adding new authorities will increase the number of users and, consequently, the number of requests. At the same time, it will also increase the overall system resources, as the new authorities will devote resources to become stakeholders of the Proxy and of their Domain blockchain. The system's performance is linearly dependent on the available resources, which means that as resources increase, the overall time decreases. Note that in both system setups the system presents zero errors per requests, due to the queuing module integration.

6. Security analysis

Threat model. We consider both internal and external attackers. Internal attackers may be compromised nodes of the HMBAC system or compromised users. Compromised nodes may attempt to illegally modify the access policies or the domain's stakeholders' set. Compromised users may attempt to bypass access control policies and gain unauthorized access. External attackers may attempt to gain unauthorized access to the system.

Assumptions. We shall assume in our analysis that the underlying software components such as the orchestration engine (Kubernetes) and the isolation mechanisms (Pods and Hashicorp Vault) are trusted. Instead of requiring a fully trusted authority, we relax our trust assumptions to a majority of trusted stakeholders for each domain. We assume that the majority of the participants in the consensus and voting protocols behave in a trusted way. We assume that the encryption and authentication mechanisms used (AES and MA-ABE) are secure (cannot be compromised by a probabilistic polynomial-time Turing machine). Finally, we assume that the user credentials are securely managed at the user side. As the main goal of HMBAC is to provide access control, we will first examine security against unauthorized access attacks and then other security characteristics of the proposed system.

6.1. *Secure data access*

The security of HMBAC controlled data access is based on several security building blocks (as detailed in Section 4.2). First, data is ABE encrypted with the keys assigned to users based on their roles, by applying the MA-ABE scheme in [6]. Then, an extra layer

doi:10.20944/preprints202212.0312.v1

of ABE encryption is performed, with an attribute key assigned to the Key Store Smart Contract (KSSC). This is implemented by applying an additional 'AND' rule, on top of the predefined encryption policy. This forces all requests to be performed via the HMBAC system; otherwise, the data retrieved by users will still be partially encrypted. The BC-side attribute keys are securely stored in a Vault and are accessible only by the KSSC.

Besides the encryption layer, the user must get authenticated by the system in order to send queries, and also by the user-side Vault to access the attribute keys, in order to decrypt the received partially encrypted data. System authentication is performed through the proxy blockchain using the Trust Management Smart Contract (TMSC). An authenticated user may then send a data access request, which in turn will be validated at the domain blockchain layer, via the Access Control Smart Contract (ACSC), in order to verify that the user has the required roles based on the access policy. The KMSC performs the required partial decryption. Finally, users need access to their attribute certificates, issued by the relevant stakeholders / authorities, to verify their roles with the ACSC[8].

To formalize our analysis of unauthorized access attacks, we use attack trees as in [53]. Attack trees [54] are a conceptual design used to describe attacks on system assets. We distinguish two types of attack nodes, *and-nodes* and *or-nodes*: the children of an and-node should all be executed to reach the goal of their parent, while any one of the children of an or-node needs to be executed to reach the goal of its parent. An attack on the system is then modeled by a multi-set of compromised nodes.

Definition [53]. Let $\mathbb{C}$ be a set of attack components of a system. An *attack* is a finite non-empty multi-set of $\mathbb{C}$ and an *attack suite* is a finite set of attacks. Denote the universe of attacks by $\mathbb{A} = \mathcal{M}^{+}(\mathbb{C})$ and the universe of attack suites by $\mathbb{S} = \mathcal{P}(\mathbb{A})$.

The attack tree for unauthorized data access attacks on HMBAC is shown in Figure 8. Our goal is to analyze all possible attack paths for an adversary, external and/or internal, to compromise the access control mechanism and gain unauthorized data access. As defined in our threat model, accessing the data in ways that are outside the HMBAC system are out of scope, e.g. accessing the data before they are ABE encrypted or before their entry into the system.

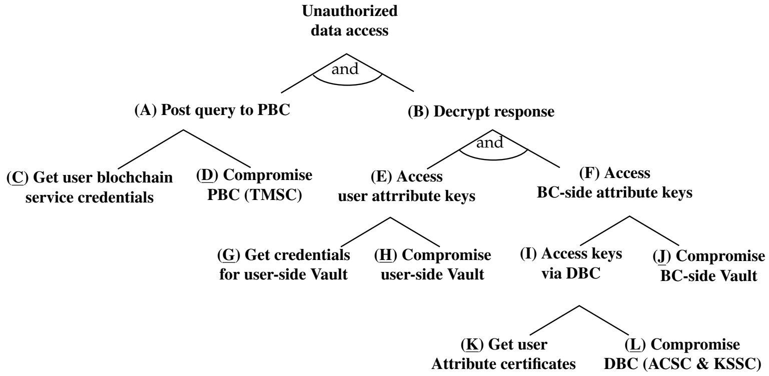
Unauthorized data access
and
(A) Post query to PBC
(B) Decrypt response
and
(C) Get user blochchain service credentials
(D) Compromise PBC (TMSC)
(E) Access user attrribute keys
(F) Access BC-side attribute keys
(G) Get credentials for user-side Vault
(H) Compromise user-side Vault
(I) Access keys via DBC
(J) Compromise BC-side Vault
(K) Get user Attribute certificates
(L) Compromise DBC (ACSC & KSSC)

Figure 8. Unauthorized data access attack tree for the HMBAC architecture

To construct the attack tree, first we observe that unauthorized data access requires an adversary to concurrently bypass the security mechanisms that: validate a data access query posted to the PBC (denoted by node A), *and* access all the attribute keys used to encrypt the data (denoted by node B). Note that despite the actual attack that may be applied to achieve the above conditions, simultaneously achieving the attack components A and B are necessary and sufficient conditions for any successful attack on unauthorized data access against an HMBAC system. Then for each level-1 node we continue our analysis of identifying all possible sets of system components that must be successfully attacked

[8] Note that the attribute certificates may also be stored in a user-side Vault for protection.

to achieve each the goal of the relevant parent node. The same holds for all nodes of the attack trees, including the leaf nodes.

Note that for all nodes, including leaf nodes, we did not examine the actual attack techniques that may be used to achieve the relevant goal. For example, for node C there exist various implementations of attacks to obtain user credentials for the HMBC service, such as phishing, spoofing, or brute force. The goal of the attack tree analysis is to exhaustively list all possible sets of necessary attack steps (i.e., concurrently compromised security components) to succeed in the attack.

For this tree, the set of identified attack components (nodes) is:

$\mathbb{C} = \{A, B, \underline{C}, \underline{D}, E, F, \underline{G}, \underline{H}, I, \underline{J}, \underline{K}, \underline{L}\}$, with seven leaf nodes (for clarity, leaf nodes are underlined).

Leaf nodes are vulnerable components that the attacker may exploit to initiate an attack. Any attack suite must contain such nodes, as well as the target node T.

We examine the attack suites of the unauthorized data access attack tree of the HMBAC, with respect to the successful attack steps required by an adversary. We consider the following cases:

Case 1. *Fully compromised user*: all user credentials (BC credentials or PBC access (C or D), user-side Vault credentials (G or H) and attribute certificates (K)) are compromised. We get the attacks: $\{\underline{C}, A\}$, $\{\underline{D}, A\}$, $\{\underline{G}, E, B\}$, $\{\underline{H}, E, B\}$ and $\{\underline{K}, I, F, B\}$, that when combined give us the attack suites:

$$\begin{aligned}
S_{1cgk} &= \{\underline{C}, A, \underline{G}, E, \underline{K}, I, F, B, T\}, \\
S_{1dgk} &= \{\underline{D}, A, \underline{G}, E, \underline{K}, I, F, B, T\}, \\
S_{1chk} &= \{\underline{C}, A, \underline{H}, E, \underline{K}, I, F, B, T\}, \\
S_{1dhk} &= \{\underline{D}, A, \underline{H}, E, \underline{K}, I, F, B, T\}.
\end{aligned}$$

The attacker will then be able to post to the system all queries available to the target user. However, this attack does not leak the data from other users.

Case 2. *Partially compromised user*: at least one of the required user credentials C, D, G, H and K is secure. In this case, from the attacks: $\{\underline{C}, A, \underline{G}, E\}$, $\{\underline{D}, A, \underline{G}, E\}$, $\{\underline{C}, A, \underline{H}, E\}$, $\{\underline{D}, A, \underline{H}, E\}$, and $\{\underline{K}, I, F\}$, $\{\underline{L}, I, F\}$, $\{\underline{J}, F\}$, we get the attack suites:

$$\begin{aligned}
S_{2cgk} &= \{\underline{C}, A, \underline{G}, E, \underline{K}, I, F, B, T\}, \\
S_{2cgl} &= \{\underline{C}, A, \underline{G}, E, \underline{L}, I, F, B, T\}, \\
S_{2cgj} &= \{\underline{C}, A, \underline{G}, E, \underline{J}, F, B, T\}, \\
S_{2dgk} &= \{\underline{D}, A, \underline{G}, E, \underline{K}, I, F, B, T\}, \\
S_{2dgl} &= \{\underline{D}, A, \underline{G}, E, \underline{L}, I, F, B, T\}, \\
S_{2dgj} &= \{\underline{D}, A, \underline{G}, E, \underline{J}, F, B, T\}, \\
S_{2chk} &= \{\underline{C}, A, \underline{H}, E, \underline{K}, I, F, B, T\}, \\
S_{2chl} &= \{\underline{C}, A, \underline{H}, E, \underline{L}, I, F, B, T\}, \\
S_{2chj} &= \{\underline{C}, A, \underline{H}, E, \underline{J}, F, B, T\}, \\
S_{2dhk} &= \{\underline{D}, A, \underline{H}, E, \underline{K}, I, F, B, T\}. \\
S_{2dhl} &= \{\underline{D}, A, \underline{H}, E, \underline{L}, I, F, B, T\}, \\
S_{2dhj} &= \{\underline{D}, A, \underline{H}, E, \underline{J}, F, B, T\}.
\end{aligned}$$

Again, these attacks only affect the data of the compromised users.

Case 3. *Fully compromised PBC (D) and DBC (L).* Here unauthorized queries get posted due to a compromised Proxy BC (bypassing the TMSC), while access to the BC-side keys assumes a compromised domain BC (bypassing the ACSC control and utilizing the BC-side attribute keys via the KSSC). However, a successful attack suite requires additionally access to the user attribute keys, either by compromising the user-side Vault (H) or by getting the user credentials (G). We get the attack suites:

$$\begin{aligned}
S_{3dhl} &= \{\underline{D}, A, \underline{H}, E, \underline{L}, I, F, B, T\}, \\
S_{3dgl} &= \{\underline{D}, A, \underline{G}, E, \underline{L}, I, F, B, T\}.
\end{aligned}$$

doi:10.20944/preprints202212.0312.v1

Case 4. *Fully compromised Vault.* Here both the user- side and BC-side Vaults (H and J) are compromised. Again, a successful attack requires additionally a partially compromised user (C) or Proxy BC (D). We get the attack suites:

$$\begin{aligned} S_{4chj} &= \{\underline{C}, A, \underline{H}, E, \underline{J}, F, B, T\}, \\ S_{4dhj} &= \{\underline{D}, A, \underline{H}, E, \underline{J}, F, B, T\}. \end{aligned}$$

Case 5. *All entities partially compromised.* Here the user credentials/certificates (C, K), blockchains (D, L) and Vault storage (G, H, J), are all partially compromised. We get the attack suites:

$$\begin{aligned} S_{5chl} &= \{\underline{C}, A, \underline{H}, E, \underline{L}, I, F, B, T\}, \\ S_{5dgj} &= \{\underline{D}, A, \underline{G}, E, \underline{J}, F, B, T\}, \\ S_{5dhk} &= \{\underline{D}, A, \underline{H}, E, \underline{K}, I, F, B, T\}. \end{aligned}$$

We now have:

Proposition 1. *Compromised user credentials (either fully or partially) cannot affect the data access of other users.*

Proof. This follows directly from Cases 1 and 2. □

Proposition 2. *The system can resist unauthorized data access even if both the proxy and the domain blockchains are compromised, provided that the user attribute keys are secure.*

Proof. This follows directly from Case 3. □

Proposition 3. *The system can resist unauthorized data access if at least one of the system entities (users, blockchains, key Vaults) are secure.*

Proof. This follows directly from Cases 4 and 5. □

6.2. Secure blockchain management

The security of critical management decisions that could compromise the system's security relies on: (i) the voting mechanism implemented at the Proxy blockchain, (ii) the blockchain consensus mechanism, (iii) the transaction replication implemented by all the blockchains, and (iv) the execution isolation supported by the use of Kubernetes and independently managed Pods.

As explained in Section 5.1 the voting mechanism, implemented by the PSC, enables the stakeholders to reach management decisions. Any stakeholder may start an election. Voters' eligibilty and vote integrity is ensured, since the private key of a stakeholder is required to sign a vote for an election. Different thresholds and eligible voters can be defined for different elections.

The blockchain consensus mechanism is also related to secure system management. Since smart contracts in both blockchain layers implement critical functionality of the system, modifying those smart contracts either at the PBC or at the DBCs could compromise the security of policy enforcement. However, as the smart contracts are implemented in the initial blocks of each blockchain, their integrity is strongly protected.

Since the underlying consensus mechanism of Fabric (Raft) does not support Byzantine tolerance, a malicious leader might attempt to forge the blockchain(s) logic by adding modified smart contracts, e.g. to compromise the access policy. However, such an attack would be easily detected by the other stakeholders because of the blockchain replication mechanism and the lack of integrity (valid signatures by the stakeholders' majority) of the modified smart contracts. Finally, the encapsulation of all the distributed components in replicated independent Pods, executed by different stakeholders and orchestrated by Kubernetes, also protects system integrity.

6.3. Secure key storage/management

The use of Hashicorp Vault provides secure key storage. For each DBC, an independent vault instance is used to store and securely access the domain's attribute key. In addition, users may also deploy vault instances to protect their attribute keys and attribute certificates. Finally, certificate management at the stakeholder level is implemented by independent instances of Hyperledger Fabric CA running on different Pods. These are accessible by the TMSC through encrypted and authenticated Kubernetes ports.

7. Discussion and Conclusions

Hierarchical multichains, when coupled with Attribute Based Encryption, provide a flexible and secure distributed access controls mechanism for multi-domain, multi-authority environments. Its modular architecture supports various properties of blockchains, such as interoperabilty, by providing a single point of access for multiple domains and single source of truth, via block-chain replication and integrity. The use of a hierarchical structure supports fine-grained access control and flexible management. At the same time, the integration of distributed MA-ABE enables the combined use of credentials issued by multiple authorities without introducing a high management overhead. In this paper, we have defined HMBAC, a novel access control model along with Janus, an actual implementation of an HMBAC system, and we have analyzed the security and efficiency of our implementation.

In the future, we intend to explore the integration of different consensus mechanisms to extend the applicability in environments that require strong Byzantine tolerance. In addition, since user credential management is at the stakeholder level and is managed outside our system, it is possible that inefficient user credential management can affect the overall efficiency of the system. Possible ways to minimize this risk can also be examined in the future.

Author Contributions: Conceptualization, Vangelis Malamas and Panayiotis Kotzanikolaou; Data curation, George Palaiologos; Formal analysis, Vangelis Malamas, Panayiotis Kotzanikolaou and Mike Burmester; Funding acquisition, Panayiotis Kotzanikolaou; Investigation, George Palaiologos and Dimitris Glynos; Methodology, Vangelis Malamas, George Palaiologos, Panayiotis Kotzanikolaou, Mike Burmester and Dimitris Glynos; Project administration, Dimitris Glynos; Resources, Vangelis Malamas and George Palaiologos; Software, Vangelis Malamas and George Palaiologos; Validation, Panayiotis Kotzanikolaou, Mike Burmester and Dimitris Glynos; Visualization, Vangelis Malamas and George Palaiologos; Writing – original draft, Vangelis Malamas, George Palaiologos and Panayiotis Kotzanikolaou; Writing – review editing, Vangelis Malamas, George Palaiologos, Panayiotis Kotzanikolaou, Mike Burmester and Dimitris Glynos. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been co - financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (project code:T1EDK-01958 and T2EDK-02836).

Institutional Review Board Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Statista. Volume of data created, captured, copied, and consumed worldwide from 2010 to 2025. Technical report, accessed, January 2022.
2. Zhang, P.; Chen, Z.; Liang, K.; Wang, S.; Wang, T. A cloud-based access control scheme with user revocation and attribute update. In Proceedings of the Australasian Conference on Information Security and Privacy. Springer, 2016, pp. 525–540.
3. Lo, C.C.; Huang, C.C.; Ku, J. A cooperative intrusion detection system framework for cloud computing networks. In Proceedings of the 2010 39th International Conference on Parallel Processing Workshops. IEEE, 2010, pp. 280–284.
4. Li, J.; Chen, X.; Chow, S.S.; Huang, Q.; Wong, D.S.; Liu, Z. Multi-authority fine-grained access control with accountability and its application in cloud. *Journal of Network and Computer Applications* **2018**, *112*, 89–96.
5. Malamas, V.; Palaiologos, G.; Kotzanikolaou, P.; Burmester, M.; Glynos, D. Janus. https://github.com/CENSUS/Janus, 2022.

6. Lewko, A.; Waters, B. Decentralizing attribute-based encryption. In Proceedings of the Annual international conference on the theory and applications of cryptographic techniques. Springer, 2011, pp. 568–588.
7. Malamas, V.; Kotzanikolaou, P.; Dasaklis, T.K.; Burmester, M. A hierarchical multi blockchain for fine grained access to medical data. *IEEE Access* **2020**, *8*, 134393–134412.
8. Al Nuaimi, K.; Mohamed, N.; Al Nuaimi, M.; Al-Jaroodi, J. A survey of load balancing in cloud computing: Challenges and algorithms. In Proceedings of the 2012 second symposium on network cloud computing and applications. IEEE, 2012, pp. 137–142.
9. Andersen, M.P.; Kumar, S.; AbdelBaky, M.; Fierro, G.; Kolb, J.; Kim, H.S.; Culler, D.E.; Popa, R.A. {WAVE}: A decentralized authorization framework with transitive delegation. In Proceedings of the 28th {USENIX} Security Symposium ({USENIX} Security 19), 2019, pp. 1375–1392.
10. Shafagh, H.; Burkhalter, L.; Ratnasamy, S.; Hithnawi, A. Droplet: Decentralized Authorization and Access Control for Encrypted Data Streams. In Proceedings of the 29th {USENIX} Security Symposium ({USENIX} Security 20), 2020, pp. 2469–2486.
11. Rajput, A.R.; Li, Q.; Ahvanooey, M.T.; Masood, I. EACMS: Emergency access control management system for personal health record based on blockchain. *IEEE Access* **2019**, *7*, 84304–84317.
12. Shahraki, A.S.; Rudolph, C.; Grobler, M. A dynamic access control policy model for sharing of healthcare data in multiple domains. In Proceedings of the 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). IEEE, 2019, pp. 618–625.
13. Miao, Y.; Liu, X.; Choo, K.K.R.; Deng, R.H.; Li, J.; Li, H.; Ma, J. Privacy-preserving attribute-based keyword search in shared multi-owner setting. *IEEE Transactions on Dependable and Secure Computing* **2019**.
14. Andersen, M.; Kumar, S. "WAVE". https://github.com/immesys/wave, 2019.
15. "Dropbox Compression". https://github.com/dropbox/lepton, 2019.
16. Xu, Y.; Dong, X.; Shen, J. Multi-authority attribute-based encryption supporting hierarchal access policy and range policy. In Proceedings of the 2020 International Conference on Computer Communication and Network Security (CCNS). IEEE, 2020, pp. 81–86.
17. Xiao, M.; Hu, X. Multi-authority attribute-based encryption access control scheme in wireless body area network. In Proceedings of the 2018 3rd International Conference on Information Systems Engineering (ICISE). IEEE, 2018, pp. 39–45.
18. Zhang, Z.; Zhou, S. A decentralized strongly secure attribute-based encryption and authentication scheme for distributed Internet of Mobile Things. *Computer Networks* **2021**, *201*, 108553.
19. Riad, K.; Huang, T.; Ke, L. A dynamic and hierarchical access control for IoT in multi-authority cloud storage. *Journal of Network and Computer Applications* **2020**, *160*, 102633.
20. Bai, L.; Fan, K.; Bai, Y.; Cheng, X.; Li, H.; Yang, Y. Cross-domain access control based on trusted third-party and attribute mapping center. *Journal of Systems Architecture* **2021**, *116*, 101957.
21. Li, Q.; Zhu, H.; Xiong, J.; Mo, R.; Ying, Z.; Wang, H. Fine-grained multi-authority access control in IoT-enabled mHealth. *Annals of Telecommunications* **2019**, *74*, 389–400.
22. Sarma, R.; Kumar, C.; Barbhuiya, F.A. MACFI: A multi-authority access control scheme with efficient ciphertext and secret key size for fog-enhanced IoT. *Journal of Systems Architecture* **2022**, *123*, 102347.
23. Guo, H.; Meamari, E.; Shen, C.C. Multi-authority attribute-based access control with smart contract. In Proceedings of the Proceedings of the 2019 international conference on blockchain technology, 2019, pp. 6–11.
24. Das, S.; Namasudra, S. Multi-Authority CP-ABE-Based Access Control Model for IoT-Enabled Healthcare Infrastructure. *IEEE Transactions on Industrial Informatics* **2022**.
25. Liu, C.; Xiang, F.; Sun, Z. Multiauthority Attribute-Based Access Control for Supply Chain Information Sharing in Blockchain. *Security and Communication Networks* **2022**, *2022*.
26. Banerjee, S.; Bera, B.; Das, A.K.; Chattopadhyay, S.; Khan, M.K.; Rodrigues, J.J. Private blockchain-envisioned multi-authority CP-ABE-based user access control scheme in IIoT. *Computer Communications* **2021**, *169*, 99–113.
27. Xu, Q.; Tan, C.; Fan, Z.; Zhu, W.; Xiao, Y.; Cheng, F. Secure multi-authority data access control scheme in cloud storage system based on attribute-based signcryption. *IEEE Access* **2018**, *6*, 34051–34074.
28. Li, H.; Pei, L.; Liao, D.; Chen, S.; Zhang, M.; Xu, D. FADB: A fine-grained access control scheme for VANET data based on blockchain. *IEEE Access* **2020**, *8*, 85190–85203.
29. Malamas, V.; Dasaklis, T.; Kotzanikolaou, P.; Burmester, M.; Katsikas, S. A forensics-by-design management framework for medical devices based on blockchain. In Proceedings of the 2019 IEEE world congress on services (SERVICES). IEEE, 2019, Vol. 2642, pp. 35–40.
30. Wang, G.; Liu, Q.; Wu, J. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In Proceedings of the Proceedings of the 17th ACM conference on Computer and communications security, 2010, pp. 735–737.
31. Wan, Z.; Deng, R.H.; et al. HASBE: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing. *IEEE transactions on information forensics and security* **2011**, *7*, 743–754.
32. Ali, M.; Mohajeri, J.; Sadeghi, M.R.; Liu, X. A fully distributed hierarchical attribute-based encryption scheme. *Theoretical Computer Science* **2020**, *815*, 25–46.

33. Gai, K.; Guo, J.; Zhu, L.; Yu, S. Blockchain meets cloud computing: A survey. *IEEE Communications Surveys & Tutorials* **2020**, *22*, 2009–2030.
34. Riabi, I.; Ayed, H.K.B.; Saidane, L.A. A survey on Blockchain based access control for Internet of Things. In Proceedings of the 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC). IEEE, 2019, pp. 502–507.
35. Casino, F.; Dasaklis, T.K.; Patsakis, C. A systematic literature review of blockchain-based applications: Current status, classification and open issues. *Telematics and informatics* **2019**, *36*, 55–81.
36. Sukhodolskiy, I.; Zapechnikov, S. A blockchain-based access control system for cloud storage. In Proceedings of the 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). IEEE, 2018, pp. 1575–1578.
37. Wang, S.; Zhang, Y.; Zhang, Y. A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems. *Ieee Access* **2018**, *6*, 38437–38450.
38. Yang, C.; Tan, L.; Shi, N.; Xu, B.; Cao, Y.; Yu, K. AuthPrivacyChain: A blockchain-based access control framework with privacy protection in cloud. *IEEE Access* **2020**, *8*, 70604–70615.
39. Chase, M. Multi-authority attribute based encryption. In Proceedings of the Theory of cryptography conference. Springer, 2007, pp. 515–534.
40. Qian, H.; Li, J.; Zhang, Y.; Han, J. Privacy-preserving personal health record using multi-authority attribute-based encryption with revocation. *International Journal of Information Security* **2015**, *14*, 487–497.
41. Rouselakis, Y.; Waters, B. Efficient statically-secure large-universe multi-authority attribute-based encryption. In Proceedings of the International Conference on Financial Cryptography and Data Security. Springer, 2015, pp. 315–332.
42. Ramesh, D.; Priya, R. Multi-authority scheme based CP-ABE with attribute revocation for cloud data storage. In Proceedings of the 2016 International Conference on Microelectronics, Computing and Communications (MicroCom). IEEE, 2016, pp. 1–4.
43. Zhang, Z.; Li, C.; Gupta, B.B.; Niu, D. Efficient compressed ciphertext length scheme using multi-authority CP-ABE for hierarchical attributes. *IEEE Access* **2018**, *6*, 38273–38284.
44. Li, J.; Zhang, R.; Lu, Y.; Han, J.; Zhang, Y.; Zhang, W.; Dong, X. Multiauthority Attribute-Based Encryption for Assuring Data Deletion. *IEEE Systems Journal* **2022**.
45. Oktian, Y.E.; Lee, S.G.; Lee, H.J. Hierarchical multi-blockchain architecture for scalable internet of things environment. *Electronics* **2020**, *9*, 1050.
46. Lee, N.Y. Hierarchical Multi-Blockchain System for Parallel Computation in Cryptocurrency Transfers and Smart Contracts. *Applied Sciences* **2021**, *11*, 10173.
47. Tong, W.; Dong, X.; Shen, Y.; Jiang, X. A Hierarchical Sharding Protocol for Multi-Domain IoT Blockchains. In Proceedings of the ICC 2019 - 2019 IEEE International Conference on Communications (ICC), 2019, pp. 1–6. https://doi.org/10.1109/ICC.2019.8761147.
48. Sim, S.H.; Jeong, Y.S. Multi-Blockchain-Based IoT Data Processing Techniques to Ensure the Integrity of IoT Data in AIoT Edge Computing Environments. *Sensors* **2021**, *21*, 3515.
49. Ma, M.; Shi, G.; Li, F. Privacy-oriented blockchain-based distributed key management architecture for hierarchical access control in the IoT scenario. *IEEE access* **2019**, *7*, 34045–34059.
50. Chang, J.; Ni, J.; Xiao, J.; Dai, X.; Jin, H. SynergyChain: A Multichain-based Data Sharing Framework with Hierarchical Access Control. *IEEE Internet of Things Journal* **2021**.
51. Tao, Q.; Cui, X.; Huang, X.; Leigh, A.M.; Gu, H. Food safety supervision system based on hierarchical multi-domain blockchain network. *IEEE Access* **2019**, *7*, 51817–51826.
52. Gupta, M.; Patwa, F.; Sandhu, R. An attribute-based access control model for secure big data processing in hadoop ecosystem. In Proceedings of the Proceedings of the Third ACM Workshop on Attribute-Based Access Control, 2018, pp. 13–24.
53. Mauw, S.; Oostdijk, M. Foundations of attack trees. In Proceedings of the International Conference on Information Security and Cryptology. Springer, 2005, pp. 186–198.
54. Schneier, B. Attack trees. *Dr. Dobb's journal* **1999**, *24*, 21–29.