Received 5 January 2024; revised 30 January 2024; accepted 15 February 2024. Date of publication 19 February 2024; date of current version 1 March 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.3367461

Attribute-Based Management of Secure Kubernetes Cloud Bursting

MAURO FEMMINELLA [1,2] (Member, IEEE), MARTINA PALMUCCI [3], GIANLUCA REALI [1,2], AND MATTIA RENGO [1]

[1]Department of Engineering, University of Perugia, 06125 Perugia, Italy

[2]Consorzio Nazionale Interuniversitario per le Telecomunicazioni, 43124 Parma, Italy

[3]Consortium GARR, 00185 Rome, Italy

CORRESPONDING AUTHOR: M. FEMMINELLA (e-mail: mauro.femminella@unipg.it)

This work was supported in part by the European Union - NextGenerationEU under the Italian Ministry of University and Research (MUR) National Innovation Ecosystem under Grant ECS00000041 (VITALITY), and in part by the MUR Extended Partnerships under Grant PE0000001 (RESTART).

ABSTRACT In modern cloud computing, the need for flexible and scalable orchestration of services, combined with robust security measures, is paramount. In this paper, we propose an innovative approach for managing secure cloud bursting in Kubernetes, combining Attribute-Based Encryption (ABE) with Kubernetes labeling. Our model addresses the challenges of complexity, cost, and data protection compliance by leveraging both Kubernetes and ABE. We introduce an attribute-based bursting component that uses Kubernetes labels for orchestration, and an encryption component that employs ABE for data protection. This unified management model ensures data confidentiality while enabling efficient cloud bursting. Our approach combines the strengths of label-based orchestration with fine-grained encryption, providing a technologically advanced yet user-friendly solution for secure cloud bursting. We present a proof-of-concept implementation that demonstrates the feasibility and effectiveness of our model. Our approach offers a unified solution that complies with security and privacy laws while meeting the needs of contemporary cloud-based systems.

INDEX TERMS Cloud bursting, orchestration, attribute-based encryption, Kubernetes.

I. INTRODUCTION

IN RECENT years, the proliferation of virtualization and containerization technologies has led to a significant increase in the complexity of distributed systems, as cloud computing systems. As organizations strive to achieve efficient resource management and scalability, Kubernetes (K8s) [27] has emerged as the most popular solution for orchestrating resources in such complex systems. For example, it is integrated into platforms such as Amazon Elastic Kubernetes Service (EKS) [1], Google Kubernetes Engine (GKE) [2], Azure Kubernetes Service (AKS) [50], IBM Cloud Kubernetes Service [3], and Oracle Container Engine for Kubernetes (OKE) [5].

This paper aims at exploring the challenges associated with cloud bursting, which allows private cloud services to use public cloud resources when local resources are exhausted or for any other reasons. Specifically, we address the configuration issues associated with service request load management over a hybrid cloud system including both private and public components. The orchestration of such a heterogeneous system presents a number of challenges, such as optimal management of typically large volume of resources, variable operating conditions, security issues, and compliance with local regulations. In order to address these challenges, resorting to attribute-based management policies is regarded as a valid approach [9].

Attributes are intrinsic characteristics or properties related to the entities, workload, or resources to manage. They can refer to different aspects, such as computational requirements, security levels, data location, and other relevant information. Recent findings indicate that the related management challenges can be effectively addressed through the utilization of an attribute-based approach, which has been found to be preferred over the conventional role-based methods. Rather than depending solely on predefined roles, attributes can include a broader array of qualities and attributes associated with users, resources, or data. This level of flexibility, adaptability, and precision in access control renders them more suitable for scenarios characterized by a diverse, dynamic, and intricate range of access requirements [16], [62]. In particular, in this paper, we leverage the attribute-based approach to easily orchestrate load distribution and resource allocation between private and public clouds during the cloud bursting process.

Attribute-based policies can be enforced by different technologies. In this paper we make use of Kubernetes, since today it is one of the most appreciate tools for managing distributed systems, especially in the context of cloud computing. It provides flexibility though different built-in components and tools. Among them, the usage of labels and label selectors can be exploited to simplify cloud bursting operations. While Kubernetes best practices recommend that labels be assigned semantic meanings before being used [9], there is currently no standardized method for enforcing this practice. Our goal is to develop a systematic approach in the context of cloud bursting that ensures semantic meaning associated with the generic Kubernetes label concept.

Furthermore, it emerged that Kubernetes management does not suitably address all the security aspects related to data confidentiality and access controls, which are central in cloud bursting [6]. Kubernetes incorporates access management, but it requires separate configuration processes that are decoupled from the logic of the orchestrated functions. Moreover, the existing access management mechanisms in Kubernetes have certain limitations in terms of managing complex authorization scenarios and are constrained by their policy scope. Hence, these limitations are challenging for achieving comprehensive and secure resource management in the context of cloud bursting. To overcome these limitations, in this paper we propose an architectural solution to address the security challenges of cloud bursting that integrates the Kubernetes orchestration with attribute-based encryption.

In more detail, our specific contributions are as follows:

1) Association of semantic meaning with key labels, giving them the role of attributes. This approach adds context to the cloud bursting configuration and improves label comprehension.
2) Leveraging the Attribute-Based Encryption (ABE) technology, deployed through a cloud service, to improve security levels, in terms of data privacy, confidentiality, and access control, through fine-grained policies. This ABE component works seamlessly within the Kubernetes environment, aligning with attribute logic and improving overall system security.
3) Simplification and speed-up of configuration based on a unified management layer that handles holistically all attributes, including the ones directly used by Kubernetes and by the ABE module. This unification ensures transparency and ease of use for administrators, eliminating the need for separate configurations and additional harmonization functions, as well as a streamlined experience.

In synthesis, we propose a secure cloud bursting scheme that improves both efficiency and security in resource management over legacy solutions by incorporating semantic labels, cryptographic technology, and a unified attribute configuration approach.

The organization of the paper is outlined as follows. In Section II, we delve into an examination of the existing literature within the domain. Moving on to Section III, we furnish a comprehensive introduction to the foundational concepts. Our proposed methodology is detailed in Section IV. The outcomes of the validation of the proposed approach carried out through a proof of concept implementation are presented in Section V. Finally, Section VI includes our closing remarks and conclusions.

II. RELATED WORKS

A. SECURITY SERVICES FOR CLOUD SYSTEMS

When it is necessary to move data to the cloud, it is critical to ensure security and flexible, granular control over file access. This can be efficiently done through ABE. However, user revocation is a significant issue in ABE. In [47], the authors propose a ciphertext-policy ABE (CP-ABE) scheme with efficient user revocation for cloud storage system. User revocation is handled by introducing the concept of user group, with the rule of updating private keys of the users remaining in the group when any other user leaves it. In addition, since the computation cost of CP-ABE grows linearly with the complexity of the access structure, in order to mitigate it they propose to offload high computation demand to cloud service providers without leaking file content and secret keys. They prove that the proposed scheme can withstand collusion attack performed by the revoked users cooperating with the remaining ones. A similar approach, which requires the update of the unrevoked users' keys, is proposed in [66]. It is based on the use of a group manager to accomplish this task. It also applies re-encryption technology to prevent the revoked users from decrypting ciphertexts.

However, since the correctness of outsourcing computing results is difficult to guarantee, this approach often requires resorting to the blockchain technology for obtaining such guarantees [53]. Blockchain is used also in [41] to solve the key escrow problem by replacing the traditional key authority with a blockchain. Keys are generated collaboratively by users and the blockchain, thus preventing the latter from obtaining them alone. Alternatively, multi-authority solutions can be used to securely delete data in cloud [48], where data sharing policies can be a challenging issue.

The recent proposal by Chen et al., RABE-DI [33], is an ABE scheme capable of addressing a different problem, namely protecting data integrity after user revocation, ensuring better efficiency compared to state-of-the-art proposals.

The proposal in [44] is based on a different approach. It consists of an efficient and provably secure cloud data sharing scheme (ABEDS-RR) using CP-ABE. The scheme makes use of a semi-trusted proxy party to transform part of the ciphertext with a conversion key. Compared with most existing schemes, when the attributes of a user are changed, only the private key and conversion key of that user need to be modified, leaving the other users' keys and all ciphertext unchanged. This improves efficiency of attribute revocation and update. ABEDS-RR eliminates key escrow by generating keys for users jointly by the key authority and the cloud service provider. Hence, neither party can obtain the private key of the user without collusion with each other. In addition, the weight attributes are also used to enhance the expression ability of attributes. However, this approach seems to be more suitable to a resource-constrained environment.

In order to address the timely decryption of data stored in cloud servers, the authors in [32] propose a cooperative approach based on CP-ABE to decrypt the ciphertext. Semi-authorized users cooperatively perform decryption task, making the scheme very efficient in term of computing resources and storage cost.

Bera et al. [22] propose an approach named Attribute-Based verifiable Data Storage and data Retrieval Scheme (ABDSRS) for cloud environments. It employs an attribute-based online-offline mechanism, in which only authorized data owners can anonymously upload data to the cloud. In addition, a data user can perform searching operations over encrypted data by using keyword policy. ABDSRS allows a user to verify the correctness of the search results in cloud without interacting with any authority. Thus, in addition of being lightweight, provably secure, and guaranteeing fine-grained access control, it offers further functions in terms of anonymity, privacy, secure search and verification of results.

Ahuja and Mohanty [15] propose an extension of CP-ABE in order to provide shared access privileges among users and delegation of access privileges in a flexible manner, without sacrificing scalability and fine-grained access control. The proposed solution merges CP-ABE with a hierarchical structure [64] to achieve scalability by decentralizing the key issuing authority at different levels of hierarchy. In more detail, lower level users get secret keys from the users that have a higher position in the hierarchy. The scheme results to be resistant to cheating and collusion attacks.

Repetto et al. [55] proposed a methodological approach for designing and implementing heterogeneous security services for distributed systems. The framework utilizes a hybrid architecture based on Attribute-Based Access Control (ABAC), ABE, and blockchain technology to provide secure and efficient access control in decentralized and distributed environments. ABE cryptographic primitives are specifically used to extend the ABAC functions. They implement an online authorization procedure, support time-limited authorization, protect against collusion attacks, and protect user privacy. Such features had previously been investigated by Sciancalepore et al. in the paper [59].

Lu et al. [49] propose a policy-driven approach to secure data sharing using an integration of ABAC and ABE. Private data is shared in ciphertext form between edge nodes to mitigate potential security problems such as privacy leakage.

All the papers [49], [55], [59] propose integrating ABE and ABAC to improve security of existing solutions. However, none of them fits into the context of resource and service orchestration.

Finally, Chiquito et al. in [34] survey attribute-based approaches for access control to data, focusing on policy management and enforcement. The paper aims at identifying the key properties provided by ABAC and ABE that can be used to control data access to prevent leakage to unauthorized users while providing easy-to-manage policies. To achieve this goal, they identify knowledge gaps. As for ABE, the main identified limitations consist in implementation difficulty and lack of expressive and easy-to-manage access structures. Covering these gaps with pure cryptographic approaches is challenging due to the limited information and mechanisms that can be embedded in ciphertext and keys. To address these gaps, an integration with ABAC is considered with a policy decision point, in order to take advantage of flexibility and expressiveness of the ABAC policy languages. In this case, the main issue results in the technical limitations of translating more complex policies into the ABE access structures, beyond other technical weaknesses. In addition, also in this case, no direct mapping with service and resource orchestration exists.

B. CLOUD BURSTING

A number of proprietary solutions for cloud bursting are made available by vendors. However, most of them just focus on integration between on-premise Kubernetes clusters and elastic public computing services provided by the vendor, mainly dealing with configuration issues [10], [39], [67]. All these solutions, which clearly suffer from vendor lock-in issues, overlook some of the main security issues, especially related to data privacy and protection. Most of them provide basic yet effective solutions for securing the communication between on-premise and public clouds. A popular approach consists of using the TLS-based encryption of transmitted data in both user and control plane (see, e.g., [13]). An interesting add-on for hybrid deployment is represented by Aporeto [12], now integrated in the Prisma Cloud [8]. It proposes a Zero Trust security solution based on Kubernetes Network Policies and Role-Based Access Control (RBAC). Even the whitepaper by Forrester [28], which addresses security issues and best practices for Kubernetes, focuses on identity, network, and container security, neglecting other security weaknesses on data management. The key point is that Kubernetes is not secure by design, but security is supported through the collaboration across cloud-native ecosystems.

Below, we examine the approaches used by the vendors for implementing cloud bursting in Kubernetes. Virtual Kubelet [14] is an Open Source project within the Cloud

Native Computing Foundation (CNCF)'s sandbox project. It provides a Kubernetes Kubelet that masquerades a remote cloud provider, thus extending the computing capacity of the on-premise cluster as a *virtual* worker node. When a pod is scheduled onto this virtual worker node, Virtual Kubelet makes a public cloud instance available on-demand to run it. This workaround is necessary since most on-premise enterprise Kubernetes platforms do not implement nor support the capabilities of Cluster Autoscaler [11], used to dynamically add or remove worker nodes or node pools for matching the current cluster utilization. As such, manual operation for adding a worker node in a public cloud provider to an on-premise Kubernetes cluster often does not match the requirements of cloud bursting operations. Virtual Kubelet allows enforcing access control rules with RBAC, whereas security of communications depends on capabilities of public cloud providers.

A different approach is followed by KubeSlice [57], which creates a virtual cluster across multiple clusters (on-premise and public clouds). This is realized by creating logical application *slice* boundaries, which allow pods and services to seamlessly communicate. It provides centralized control management through KubeSlice Hub, and communications between remotes pods of the same slice are encrypted by the KubeSlice gateways running on different clusters. Access control is enforced via RBAC.

A fully centralized management is provided by Alibaba Cloud through its Distributed Cloud Container Platform for Kubernetes (ACK One) [67]. It is a distributed cloud container platform able to jointly control on-premise clusters, Alibaba Cloud resources, and third party resources. The management platform runs in the ACK cloud. It makes use of RBAC for access control and encrypted communications between clusters, in both control and user plane.

CloudBees Build Acceleration follows a different approach. Although it relies on centralized management of all clusters, including those on-premise and in public clouds, the Cluster Manager can be installed in *any* server. Computing clusters are called *cloudburst agents*, over which resources are deployed on demand (*agent cloud bursting* operation). Each worker in the cluster has to run an *Electric Agent* to correctly execute bursting operations. It supports access control via RBAC and TLS-based encryption between *agents*.

Finally, Anthos [7] is a solution provided by Google which can integrate multiple clusters. It is based on the GitOps concepts, which is an operational framework that takes DevOps best practices, developed for application development, such as version control, collaboration, compliance, and CI/CD, and uses them for infrastructure automation. As such, it synchronizes cluster configurations via a component named Anthos Config Management, which can manage bursting operations. Anthos makes use of RBAC for managing access control to resources. Communication security of both control and user plane is enforced by encrypted service mesh via TLS.

As a final note, commercial solutions for cloud bursting typically rely on a limited set of monitored metrics like CPU/RAM occupation, and are actionable when one or more of them exceeds user-defined thresholds. This aspect is addressed in the literature related to cloud bursting in Kubernetes, which is mainly focused on the criteria used to offload workload to public clouds [17]. This decision can be based on either heuristic procedures, or the solution of a specific optimization problem, even found by a machine learning engine, or pre-determined criteria. In any case, a portion of the *critical* workload is handled on-premise, and the other is assigned to a public cloud [18], [56]. Significant results have been achieved in task scheduling, specifically in the allocation of computing resources among various Kubernetes clusters. Although most approaches have been designed according to the edge-to-cloud offloading paradigm, many of these proposals can be applied also to cloud bursting operations between on-premise and public clouds. The most relevant proposals about Kubernetes scheduling in these multi-clusters scenarios can be classified in two main categories: a) proposals based on machines learning/artificial intelligence, used to take decisions about the replicas or the portion of workload to be offloaded to the public cloud, and b) proposals based on classic optimization algorithms. An interesting example of the first category is taken from mobile networks, in particular the edge-to-cloud deployment in 5G networks [36]. In this paper, Faticanti et al. make use of machine learning techniques to realize a proactive offloading of tasks by anticipating peak utilization of Kubernetes pods based on pattern recognition from historical data. Also reinforcement learning can be used to anticipate overload conditions and schedule further replicas in advance [20]. As for the second category, Carmona-Cejudo et al. [29] address the offloading of computing tasks in Kubernetes in a multi-tier architecture through the solution of an optimization problem. The recent survey in [43] provides a systematic overview of offloading solutions using both traditional optimization and machine learning techniques. Finally, the interested reader can find further details on different approaches in Kubernetes scheduling in the survey [30].

To sum up, a crucial step towards achieving a secure orchestration approach that covers both resource orchestration and enhanced security is the development of a solution that combines these two topics. Our contribution aims at filling the gap in achieving both aspects. This can be done by integrating the two topics through a solution that leverages Kubernetes labeling and ABE. In particular, access to resources and services is granted by using the same attributes that are used for their orchestration.

III. BACKGROUND

In this section, we illustrate three fundamental components of our proposal. The first component pertains to Ciphertext-Policy Attribute-Based Encryption (CP-ABE), which is a cryptographic primitive that allowed us to provide confidentiality of data related to services deployed in the cloud. The second component concerns *Cloud bursting*, which is a technique that allowed us to increase the available computing capacity in the primary cloud environment by resorting to public cloud resources. Finally, we discuss K8s, which is a platform designed for automating deployment, scaling, and management of containerized applications. Collectively, these three technologies form the knowledge foundation for our model.

A. CIPHERTEXT-POLICY ATTRIBUTE-BASED ENCRYPTION

ABE [58] is an extension of the public key encryption that allows for precise control over access to encrypted data. By ABE, access policies are established based on user attributes or characteristics, which are usually represented as sets of attributes instead of individual user identities. These attributes can consist of any information regarding users, including their position, membership, or clearance level. For example, only users with the correct combination of attributes that satisfy the established access policy can be allowed decrypting and accessing data. ABE provides fine-grained access control based on user attributes. Unlike other access control mechanisms, ABE ensures confidentiality of data through the use of encryption. Consequently, ABE requires less reliance on third parties data storage services. Moreover, ABE functions enables a high level of user independence, as authorized users can independently obtain access by decrypting ciphertexts once they possess the appropriate secret key. In this way, users can avoid interacting further with any policy-enforcing entities that may be unavailable at the time of an access request [63].

CP-ABE [23] is one of the two main types of ABE. In CP-ABE, access structures are integrated into the ciphertexts, with the encrypting user being in charge of defining the access structures. The Key Generation Authority (KGA) generates secret keys that are linked to specific user attributes. Decrypting users can decrypt a ciphertext only if they possess an authorized secret key. If the set of attributes associated with a secret key satisfies the access policy attached to the ciphertext, then the set is authorized.

Alternatively, in key-policy ABE (KP-ABE) [45] data are encrypted over a set of attributes and user keys allow accessing a tree which can distinguish attributes. However, since in KP-ABE data include only user attributes, anyone can access data, whereas CP-ABE have a direct control of data access. Given that our application scenario involves the association of attributes with users and access policies to cloud resources, our focus is on CP-ABE.

In spite of its advantages, ABE has the limitation of high computational complexity. In order to mitigate this limitation, a hybrid mode can be implemented using CP-ABE. In this approach, the message is initially encrypted with a randomly generated symmetric secret key. Only this session key is then encrypted using CP-ABE under the access policy. This technique aims at minimizing the computational complexity of the overall encryption process while still enabling fine-grained access control.

In conclusion, it is clear that, by leveraging the ABE technology, it is possible to improve security levels in various aspects. Specifically, ABE provides enhanced data privacy by allowing fine-grained access control based on user attributes, ensuring that only authorized users can access encrypted data. Additionally, ABE ensures data confidentiality by protecting sensitive data from unauthorized access, even when they are stored or transferred through untrusted media. Moreover, ABE attribute-based access control enables dynamic and flexible access management, allowing administrators to define complex access policies that can adapt to variable user attributes and security requirements. Deploying an ABE module as a cloud service allows its seamless interworking within Kubernetes clusters, aligning with attribute logic and improving overall system security.

B. CLOUD BURSTING

Cloud bursting is a popular strategy used to manage a significant increase in the demand for computational resources. The primary components of a cloud bursting solution include:

- a local environment, also referred to as *local* or on-premise (*on-prem*) cloud, which hosts the organization's resources;
- one or more secondary clouds, that are typically public clouds, providing on-demand, additional resources to expand the local environment when overloaded;
- a decision engine that autonomously manages cloud bursting while adhering to the company's policies.

The need of service flexibility and scalability in response to changes in cloud capacity requirements is the major driver of cloud bursting. Cloud bursting has to be executed in background, so that users do not encounter any service disruptions.

Some milestone papers [40], [42], [61] illustrate the evolution of cloud bursting. However, the solutions therein proposed became rapidly obsolete as they essentially rely on virtual machines, whilst current approaches have evolved to the use of containers and functions. Although their limitations, those pioneering papers still offer a robust logical foundation for the development of novel solutions.

Due to its capabilities in resource orchestration [24], cloud bursting strategies may include Kubernetes as a flexible and scalable solution to handle the process of expanding service deployment to the cloud as well as dynamically assigning resources in response to changing requirements [60]. A number of Web resources and technical reports clearly describe cloud bursting operations using Kubernetes (see, e.g., [37], [38]. Most of them discuss the general features that a cloud bursting solution should offer and relevant issues, which are typically related to networking and security aspects.

C. KUBERNETES

Kubernetes, also referred to as K8s, is an open-source platform for container orchestration. It was developed by Google based on its internal system Borg [27]. The aim was to simplify the deployment, scaling, and management of containerized applications. Users utilize declarative configuration files to specify the state of their application, and K8s is in charge of maintaining the desired state. In this way, by the use of Kubernetes, several management issues of complex and distributed applications have reduced significantly.

A K8s cluster is a distributed set of master and worker nodes. One or more master nodes manage the cluster's state and configuration, while worker nodes run containerized applications. Due to its flexibility, Kubernetes has emerged as the most popular and widely used orchestrator among numerous alternatives.

In addition, to ensure compliance with the user-defined cluster state, specified in the architectural YAML manifest files or *blueprint*, Kubernetes also includes a centralized management interface for managing regular orchestrator tasks. The process is divided into some steps, such as setting up the required network and storage resources, deploying service containers, monitoring their health, and replacing the compromised ones. Furthermore, Kubernetes offers automatic load balancing, service discovery, and horizontal scaling capabilities, which allows it to seamlessly route external traffic to the appropriate services within the cluster.

IV. ATTRIBUTE-BASED MANAGEMENT OF SECURE KUBERNETES CLOUD BURSTING

Our research targets the development of a service orchestration model using Kubernetes, which involves two key innovative aspects. First, we use Kubernetes labeling to design and implement an attribute-based orchestration system for cloud bursting. Labels, which are associated with services, are used to orchestrate service deployment in cluster nodes. Some nodes may have multiple labels with preferred policies to run multiple services; others may have just one label, while others may not even have any at all. This approach shapes a cloud bursting policy tailored to each service, which can make an optimal use of the local cluster resources before requesting additional paid resources in external secondary clouds. Second, we use ABE policies defined by referring to the *same* labels, as mentioned above. By using the different labels as attributes in ABE, we have improved security without adding any extra complexity and also obtained a more refined and selective cloud bursting. This approach not only provides confidentiality, but also enables seamless integration of ABE with the existing label-based orchestration system. The final result is a sophisticated orchestration system that conforms to data protection regulations and relies on attributes as the basis of its operations, which we denote Attributed-Based management of secure K8s cloud bursting.

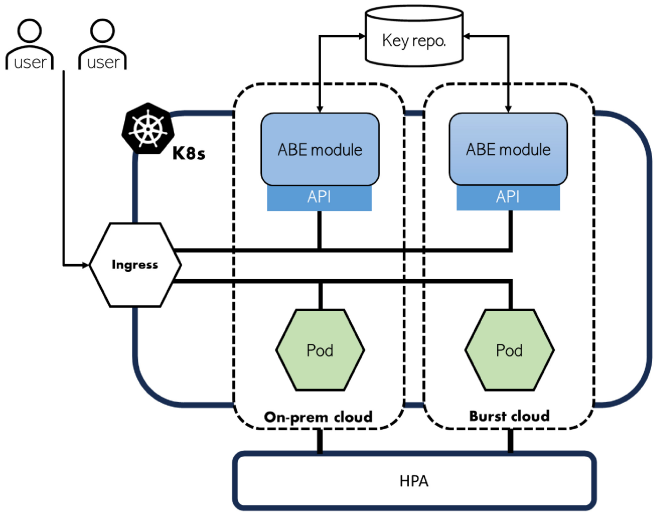


FIGURE 1. High-level diagram that illustrates the primary stakeholders and their interconnections of our model.

It is worth highlighting that K8s access control, which is the basic element of K8s security, includes both RBAC and ABAC. However, although the ABAC's flexibility is appealing in different contexts, RBAC is typically preferred over ABAC. The main reason for this is that ABAC is difficult to manage and understand in very complex distributed environments. Our proposal, which has simplicity of implementation as its key element, allows us reintroducing the flexibility of using attribute-based management in a context managed mainly with RBAC.

Below, we illustrate the structure of our proposed model, capable of achieving the aforementioned objectives.

Figure 1 depicts a high-level diagram that illustrates the primary elements of the system considered and their interconnections:

- *Users*. They access services through the K8s *Ingress* API and are represented by the user icon on the left side of Figure 1. Further, each user has a set of attributes that enable them to access only specific services, which will be discussed in what follows.
- *Clouds*. A cloud is any distributed and virtualized networked computing infrastructure that allows making use of shared pools of reconfigurable computing resources on-demand, which can be easily provisioned and withdrawn. Clouds integration within K8s provide the capability to deploy and manage containerized applications in various public cloud environments, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. By leveraging cloud-native services, Kubernetes facilitates seamless scaling, high availability, and the efficient utilization of underlying infrastructure resources.
- *Kubernetes*. As mentioned above, this open-source and flexible platform for container orchestration is efficient for automating container operations and simplifies the

orchestration of multi-container applications, ensuring that they are always running and matching the desired state. For this reason, it is a central component of our solution. Below, we outline a step-by-step workflow of how our proposed system interacts with key Kubernetes and ABE components:

- *Label Assignment and Management:* Labels, as key-value pairs, are assigned to resources (nodes and services) in Kubernetes clusters. Such labels are used not only for resource grouping and management but also play a crucial role in the attribute-based bursting and encryption process.
- *Kubernetes Metrics Server:* This component provides real-time data on resource usage by pods and nodes. It acts as the primary source of information for the HPA to make scaling decisions.
- *HorizontalPodAutoscaler (HPA).* It monitors resource usage and performance metrics of pods. Based on these metrics and the labels assigned, it makes informed decisions about scaling pod replicas up or down. This is where our attribute-based approach intersects with the Kubernetes native scaling mechanism that adjusts the number of replicas in a `Deployment` or `ReplicaSet` to meet the required performance. HPA analyzes the metrics provided by the *Metrics Server* or external monitoring systems in order to determine whether the current resource allocation aligns with the configured scaling thresholds.
- *Ingress*. It acts as an application-level load balancer that manages external access to services deployed over a cluster. Specifically, it runs an HTTP and HTTPS reverse proxy service that routes network traffic from clients to the appropriate backend services. By using Ingress APIs, complex network configurations can be managed easily, since numerous routing rules can be combined into a single resource. Ingress resources utilize an ingress controller, such as NGINX or HAProxy, to manage the routing of HTTP and HTTPS requests to backend services. Additionally, the introduction of the Ingress resource simplifies the implementation of features like SSL termination, load balancing, and name-based virtual hosting.
- *ABE Policy Enforcement:* For each service request routed through the ingress to services inside pods, the ABE module checks the attributes (aligned with the Kubernetes labels) against the defined ABE policies. This step ensures that only authorized users or services can access or interact with the data, thus maintaining data confidentiality.
- *Key Management and Data Encryption:* The Key repository plays an essential role in managing the encryption keys used in the ABE scheme. It interacts with the ABE module to provide keys for encrypting and decrypting data, thus securing the data at rest and in transit.
- *Data Flow and Access Control:* In summary, request routing within Kubernetes Ingress adheres to the K8s's label-based rules, while data access security is managed by the ABE module leveraging the label based attributes. This dual application of labeling enhances the overall operation efficiency, In fact, while the Ingress effectively manages routing of requests, ensuring they reach the correct service endpoints, the ABE module, using these same labels, determines whether a request should be granted access to the data based on the attributes of the requesting entity. This ensures that each service and the information within are accessible only under the appropriate conditions, adhering to both performance and security requirements.

- *ABE module*. This custom cloud service implements the ABE scheme in the proposed solution. It handles all the functions of an ABE scheme and manages the necessary attribute-based encryption keys. Furthermore, it establishes communication with the key repository.
- *Key repository*. It maintains the keys used to protect the volumes of services. It communicates with the ABE module.

In the subsequent sections, we describe the two primary components of our proposed solution: attribute-based bursting and encryption components.

A. ATTRIBUTE-BASED BURSTING COMPONENT

In complex systems, multiple service layers are typically involved. Their management often requires cross-cutting operations that can disrupt their strict hierarchical structure, particularly when that structure is rigidly determined by the underlying infrastructure rather than by user needs. Kubernetes uses labeling to manage such systems. Our model leverages Kubernetes labeling not just as mere identifiers, but as pivotal elements in orchestrating resource allocation and managing service demands. Labels are assigned to both services and nodes, and dictate how resources are allocated and managed within our cloud bursting ecosystem. Labeling is particularly important when it refers to autoscaling and load balancing. It allows the HPA to differentiate between resources and apply scaling policies based on the characteristics of the labels assigned to each resource.

In our proposal, the labels used in the HPA are also used as user attributes by the ABE module. In order to guarantee that services fulfill their requirements, both HPA and cloud nodes are configured with the same label values. For this purpose, it was necessary to take advantage of the concepts of *affinity* and *anti-affinity* in Kubernetes, as they influence scheduling of pods on cluster nodes:

- *Affinity* refers to the preference for scheduling pods on nodes that meet certain criteria. For example, a particular pod could have affinity to nodes with specific labels or that are located in a particular availability zone.

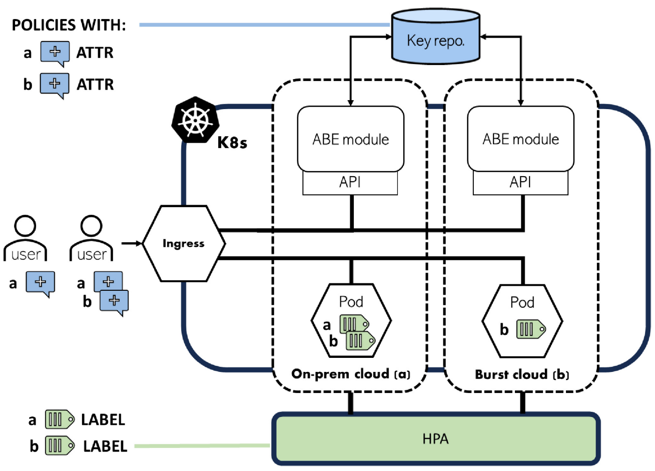


FIGURE 2. Attribute-based bursting and encryption components.

- *Anti-affinity*, on the other hand, refers to the preference for pods not to be scheduled on nodes that meet certain criteria. For example, it may be desirable that two replicas of a pod are not scheduled onto the same node for availability purposes.

Kubernetes allows setting affinity and anti-affinity values to express these types of preferences.

With reference to our system, a node with a specific label (e.g., 'On-Prem', as in Figure 2) is selected by the HPA to run the deployed services until there are no resources left in the local cloud environment, ensuring application demands are met while keeping operational expenses as low as possible.

In our proposal, the labels used for configuring Kubernetes orchestration for cloud bursting are also used as attributes used for defining ABE policies. Understanding the QoS needs for each service is crucial in any orchestration model, and in our case the Kubernetes labels are strategically used to manage both the QoS and security aspects of each service. Labels dictate not only allocation and management of resources according to QoS requirements but also form the basis of the corresponding ABE policies. This ensures seamless integration of performance management and security. For example, if two services, namely Service S_1 and Service S_2, are designed with distinct QoS and security requirements, this is reflected in their respective HPA labels and ABE policies.

This concept can be better illustrated by referring to Figure 2. In this scenario, two services are designed with variable bursting and replication requirements: service S_1, with loose requirements, and service S_2, with more strict requirements. The system setup includes one local and one external cloud, namely *On-prem cloud* and *Burst cloud*. The latter is utilized for cloud bursting. The HPA is configured to meet the specific requirements of each service, which is achieved by associating labels with each service based on their requirements. It is important to note that multiple labels may be applied to each service in practical scenarios. Label a is applied to service S_1, labels a and b to service S_2. In order to guarantee that services meet their requirements, it is necessary to configure pod affinity and anti-affinity on nodes within the cluster. In Figure 2, the nodes in the local cloud are assigned both labels, namely a and b, to enable all services to run on local nodes with affinity to every label in the system. On the other hand, nodes in Burst cloud are only assigned the label b, indicating that the nodes in Burst cloud are configured with affinity to the label b. As a result, a service that matches the required affinity only with labels assigned to local nodes can only scale out using local resources without any bursting. In contrast, a service deployment that matches affinity with both labels can be executed, duplicated and possibly leverage cloud bursting. It is worth noting that anti-affinity is not used in this example.

From a different perspective, we can say that S_1 has the strong requirement of being run only in the *on-prem cloud*, even at the cost of potentially degraded performance. By operating exclusively within the on-prem cloud environment, S_1 benefits from inherently enhanced security, as its data remain in the controlled on-prem environment. This service is tailored for operations where data security and sovereignty are paramount. Its ABE policy is accordingly structured to leverage the inherent security benefits of the on-prem environment, using a set of attributes optimized for controlled access while maintaining operational efficiency. Service S_2, on the other hand, is designed with a focus on providing superior scalability in order to keep its performance compliant with its requirements. Thus, this service is capable of operating not just on-prem, but also in external cloud environments, allowing it to scale resources dynamically based on demand. Its ABE policy is more complex, incorporating a wider range of attributes to ensure secure access across diverse environments while maintaining high availability and performance standards. This approach allows orchestration and authorization to share the same attributes and solve the issues caused by managing multiple sets of attributes. Thus, usage of labels allows implementing sophisticated and heterogeneous service policies.

As mentioned above, one of the main features of our solution is the introduction of a semantic meaning to be associated with a set of labels used in Kubernetes. In this way, these labels can be used also as ABE attributes. Clearly, it is not necessary for all labels used in Kubernetes to be used as attributes in ABE. Our system is designed to leverage such mapping onto labels whose meaning is relevant to both security and orchestration management. For example, consider a Kubernetes label called a "node-type". This label can take values like "on-prem", "burst", "a", "b" and so on. These values can be used directly as attributes in the ABE system. This approach enables the formulation of ABE security policies in accordance with the Kubernetes orchestration context. This mapping between selected Kubernetes labels and ABE attributes enables simplified management of both resource orchestration and data security, giving system administrators integrated control of these aspects.

By using the example of S_1 and S_2 above and Figure 2, we detail how our cloud-bursting architecture with ABE-based security works:

1) A user requests access to a service through the Kubernetes Ingress.
2) The Ingress controller routes the request to the appropriate service based on its label.
3) The ABE policy associated with the service label is used to authorize the user access. If the service is scaled to the burst cloud, the HPA scales the service based on its label and the current resource utilization. The ABE policy associated with the service label is used to authorize the user access.
4) The Kubernetes Ingress continues routing the user to the correct services regardless of whether it is deployed within the hybrid cloud, in the on-prem cloud or in the bust cloud.

Our cloud bursting architecture with ABE-based security offers several advantages, including:

- Performance and scalability: By dynamically scaling resources to the burst cloud, our architecture can improve performance and scalability of applications with fluctuating workloads.
- Enhanced security: ABE policies enable fine-grained access control to services, even across multiple clouds, protecting sensitive data from unauthorized access.
- Simplified management: By leveraging joint labeling for both orchestration and authorization, our architecture simplifies management and reduces complexity.
- Increased flexibility: Labels provide a versatile mechanism for implementing sophisticated and heterogeneous service policies, meeting the diverse needs of modern cloud-based applications.

B. ATTRIBUTE-BASED ENCRYPTION COMPONENT

In the previous sections, we mentioned that our objective is not only to propose a unified orchestration, but also to offer a solution able to address some security issues, such as confidentiality, access control, and compliance to General Data Protection Regulation (GDPR) rules [35]. For example, when bursting is carried out using the infrastructure of a public cloud provider, it is important to protect user data, in order to avoid privacy issues for those services that require them. In this regard, ABE natively supports the decoupling of encryption keys from third parties (i.e., the infrastructure provider), thus ensuring that only users with the right attributes can access the protected service data, even if hosted on public clouds and regardless of local regulations that may affect the cloud service provider policies. Thus, the joint management of labels for both security and orchestration management can help mitigating confidentiality leakage for critical data. In addition, since ABE allows accessing contents through policies based on the owned attributes and not on the identity, the proposed approach can be used not only to ensure confidentiality, but also to protect anonymity while accessing data outside the private cloud deployment. The level of this protection depends on several factors, such as user population, number of attributes, derived policies, and so on.

1) HYBRID CP-ABE FOR SECURE AND EFFICIENT CLOUD VOLUME ENCRYPTION

To address this objective, we propose utilizing CP-ABE in a hybrid mode in order to secure sensitive volumes. CP-ABE offers advantages such as fine-grained access control and adaptable policies. The hybrid mode balances security and efficiency by combining asymmetric and symmetric encryption, providing robust data protection while minimizing computational overhead.

In simple terms, the process illustrated below is followed. Managers of a specific service encrypt the entire volume or a portion of the associated volume. This encryption employs a symmetric cipher, with the encryption key referred to as the data encryption key (DEK). The service provider then securely stores the DEK in a designated key repository. Importantly, the DEK is not stored in plain text; rather, it is encrypted using CP-ABE. The policy selected during DEK encryption dictates who can decrypt the key, thereby obtaining access to the corresponding service.

On the user side, system participants possess various attributes. Utilizing these attributes, the system provides them with distinct decryption keys — referred to as *user keys* — enabling access to specific portions of the key repository. Essentially, only users possessing attributes that satisfy the encryption policy are authorized to access the DEK and subsequently retrieve data from the volumes.

The sequence diagram of ABE setup and operation is depicted in Figure 3. This protocol facilitates secure communication and information exchange among users, the ABE module, the key repository, and services. It serves as a safeguard, shielding users from untrusted service solutions. In fact, the only entity with which the user interacts and shares sensitive information is the ABE module — a trusted party.

Users must recover the DEKs stored in the key repository to access encrypted data within service volumes. This process consists of three phases: service setup, user registration, and service data access.

During the *service setup* phase, the service first generates a DEK, encrypts it using ABE, and stores it securely in the Key Repository within the ABE cloud service. Concurrently, the *user setup* phase consists of user registration, during which the user provides the attributes obtained from a system authority. The service may perform some operational actions that we call generic AAA actions, indicated as Admission control in the figure. They are additional security related actions that do not fall in the scope of our proposal, but often used in operation. For example, some preliminary traffic filtering can be used to mitigate potential attacks, such as

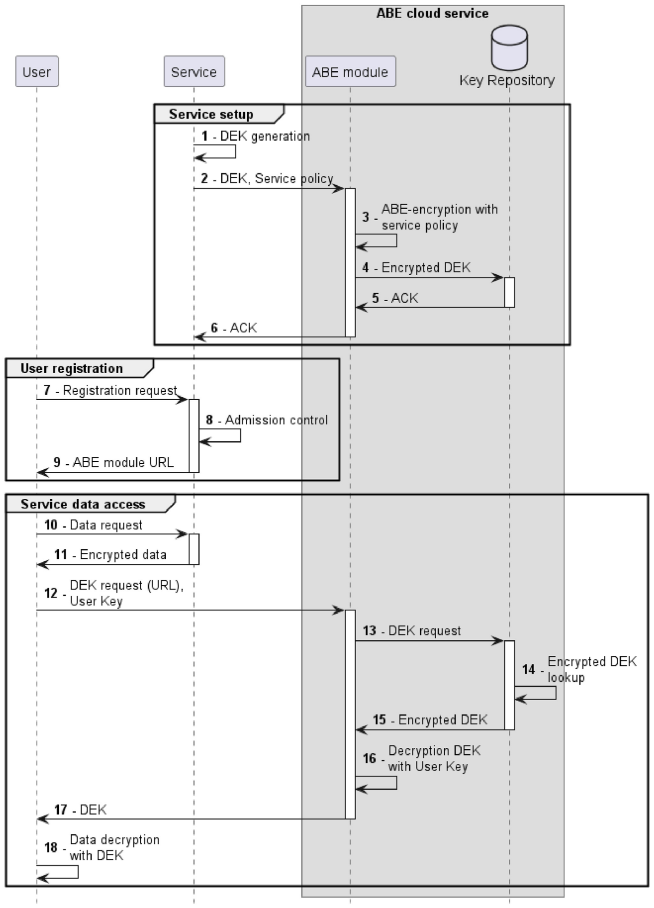


FIGURE 3. **Sequence diagram of ABE setup and operation.**

Denial of Service (DoS). Then, the service sends the ABE module URL to the user for subsequent exchanges.

Assuming the service has encrypted its data with a DEK, the subsequent *service data access* phase follows. In this phase, when a user requests data, the service provides encrypted data. Having a valid User Key, obtained during the User setup phase, the user can request the DEK from the Key Repository using the ABE module URL received earlier. The ABE module decrypts the DEK, allowing the user to decrypt and access the requested data. Once successfully decrypted, users can proceed retrieving data from services. This integrated process ensures secure and controlled data access within the cloud environment.

Moreover, cost-effectiveness is achieved by predominantly employing symmetric encryption for data encryption, which is notably faster and less computationally demanding than asymmetric encryption.

2) UNIFIED ATTRIBUTE MATCHING: SIMPLIFYING CONFIGURATION AND MANAGEMENT OF CLOUD SECURITY

Referring back to Figure 2, we can delve into another concept for further clarification. Let us turn our attention to the ABE module, the key repository, and the attribute values assigned to users. As previously mentioned, the DEKs encrypted with a specific policy find their place in the key repository. In accordance with our proposed approach, these policies are not composed of arbitrary attributes; instead, they consist of specific attributes values that match with the same set of values of the labels in the HPA.

Examining Figure 2, we observe that the policies within the key repository are formed exclusively with attributes a and b. Notably, these same attributes assigned to users also possess values within the same set. Remarkably, these are the same values employed earlier for the HPA labels.

This concept is a pivotal aspect of our proposal. Matching of label values with attribute values is essential, since this matching streamlines and expedites configuration and management. By introducing a unified management layer that comprehensively handles all attributes, including those from Kubernetes, and integrating it with the ABE module, we achieve transparency and ease of administration. This integration eliminates the need for distinct configurations and additional harmonization processes, resulting in a streamlined user experience.

3) EXPANDING POLICY EXPRESSIVENESS WITH ABE

The third key concept is related to the type of policy that can be expressed using ABE. In contrast to simple label-based policies, the ABE schema can be selected to articulate policies with greater expressive capacity. This enables the inclusion of logical expressions involving conjunctions, disjunctions, and negations. This expanded capability is outlined in the taxonomy of ABE schemes presented in [63].

The most sophisticated policies fall under the category of Non-Monotone Span Program (NMSP). For example, basic policies could be represented as follows:

- Policy $= a$
- Policy $= b$

Alternatively, more complex policies can encompass conjunctions, disjunctions, and/or negations:

- Policy $=$ NOT a
- Policy $= a$ OR b
- Policy $= a$ AND b
- Policy $= a$ OR NOT b
- Policy $=$ NOT a AND b

and so forth, encompassing all feasible combinations. These expressive capabilities enable ABE to accommodate a diverse range of policies, enhancing its versatility and applicability.

4) THE OPERATIONAL ASPECTS OF THE ABE SCHEME FOR CLOUD SECURITY

Finally, we outline the ABE scheme selected for our experiments. We opted for the well-known scheme proposed by Lewko and Waters [46]. Our decision was guided by the scheme flexibility in adapting to the dynamic nature of cloud environments, allowing for the integration of multiple authorities in a decentralized system without requiring coordination between them. Nonetheless, it is

TABLE 1. Symbols.

Symbol	Description
λ	Security parameter
GP	Global parameters
SK	Authority's Private Key
PK	Authority's Public Key
GID	User's Global identity
i	i-th attribute
$\mathrm{K}_{i,\mathrm{GID}}$	User's Secret Key share of i attribute
M	Plaintext
(A, ρ)	Access matrix (Policy)
CT	Ciphertext

worth noting that this choice is not a constraint, and any other ABE scheme can be adapted in our system and procedures.

Lewko and Waters' Multi-Authority ABE scheme [46] eliminates the need of a central authority. Decentralizing key generation and issuance allows users to encrypt data by using boolean equations based on attributes received from different authorities. The key idea presented in [46] consists of using a hash function $H(\cdot)$ on the user's global identity, GID. This allows handling collusion resistance across multiple key generations by different authorities. In more detail, $H(\cdot)$ hashes each identity to a bilinear group element (for details see Appendix A and [25], [26]). $H(\mathrm{GID})$ is used to link together keys issued to the same user by different authorities [31]. Although this way is more challenging than the single authority case, it has the great advantage of avoiding relying on a single entity.

We now consider the distinct functions constituting a multi-authority scheme, as demonstrated by Lewko and Waters [46]. The following description is not a detailed explanation of all the mathematical details of the LW-ABE scheme, which are available in the original document, but it serves to contextualize the functions of this scheme in the proposed application environment. Figure 4 shows a visual representation of the key functions of ABE within the context of our application although, for the sake of clarity, the figure includes a single authority. It is essential to note that multiple authorities can also be involved in providing users with private key. We clarify this concept below. Additionally, Table 1 reports the symbols utilized in what follows and their meaning. The Appendices contain the essential mathematical background to understand the basic operation of the functions used.

The fundamental function initiating the process is the global system setup. This step establishes the global parameters (GP), serving as the basis for the subsequent operations.

Step 0: Global Setup$(\lambda) \rightarrow$ GP. The global setup algorithm receives the λ security parameter as input and outputs global parameters for the system. In the global setup, a bilinear group G of order $N = p_1 p_2 p_3$ is selected, where N is the product of 3 primes. The output GP consists of N and a generator g_1 of G_{p_1}.[1] In addition, the hash function $H(\cdot)$ that maps GID to elements of G is published.

Once the global parameters are established, one or more authorities can be set up. Each of them is responsible for an attribute or a set of attributes.

Step 1: Authority Setup(GP) $\rightarrow$ SK, PK. Each authority runs the authority setup algorithm with GP as input to produce its own secret key (SK) and public key (PK) pair. For each attribute i belonging to an authority, it selects two random numbers $\alpha_i, y_i \in \mathbb{Z}_N$ and publishes PK $= \{e(g_1, g_1)^{\alpha_i}, g_1^{y_i} \forall i\}$, keeping secret SK $= \{\alpha_i, y_i \forall i\}$. The authorities independently generate portions of a user private key in a multi-authority system. Each user holds attributes from distinct authorities, and the absence of the need for coordination between them is a noteworthy characteristic.

The user global identity (GID) ties together the portions of the secret key generated by different authorities [31]. This is related to two crucial functions: (1) ensures that a user must use all the pieces of the private key together for successful decryption, (2) prevents users from exchanging key portions, thus mitigating collusion attacks [63].

Step 2: KeyGen(GID, GP, i, SK) $\rightarrow \mathrm{K}_{i,\mathrm{GID}}$. The key generation algorithm receives as inputs an identity GID, the global parameters, an attribute i belonging to some authority, and the secret key SK for this authority. Then, it produces a key $\mathrm{K}_{i,\mathrm{GID}} = g_1^{\alpha_i} H(\mathrm{GID})^{y_i}$ for this attribute-identity pair.

A data owner can encrypt a message or plaintext M by using a policy expressed as a (A, ρ)-pair. A is an $n \times l$ matrix, defined as the share-generating matrix for the linear secret-sharing scheme, which is the mathematical theory underlying the overall machinery.[2] ρ denotes a mapping function associating each row of A with a specific attribute. In our application, which uses ABE in hybrid mode, the plaintext M assumes the role of the DEK, and the data owner, represented by the service, can encrypt content using a DEK. This encrypted content is then made exclusively available to specific users who meet the criteria outlined in the selected policy.

Step 3: Encrypt(M, (A, ρ), GP, {PK}) $\rightarrow$ CT. The encryption algorithm takes in a message M, an access matrix (A, ρ), the set of public keys for the relevant authorities, and the global parameters GP. It outputs a ciphertext CT. In more detail, the encrypter selects a random $s \in \mathbb{Z}_N$ and a random vector $v \in \mathbb{Z}_N$ with s as its first entry. We let $\lambda_x = A_x \cdot v$, where A_x is row x of A. It also selects a random vector $w \in \mathbb{Z}_N$ with 0 as its first entry. We let $\omega_x = A_x \cdot w$. For each row A_x of A, it selects a random $r_x \in \mathbb{Z}_N$. The ciphertext is computed as:

- $C_0 = Me(g_1, g_1)^s$,
- $C_{1,x} = e(g_1, g_1)^{\lambda_x} e(g_1, g_1)^{\alpha_{\rho(x)} r_x} \quad \forall x$,
- $C_{2,x} = g_1^{r_x} \quad \forall x$,
- $C_{3,x} = g_1^{y_{\rho(x)} r_x} g_1^{\omega_x} \quad \forall x$.

1. The interested reader can find details and references about bilinear groups in Appendix A.

2. The interested reader can find the basic concepts on the underlying theory of linear secret-sharing schemes in the Appendix B.

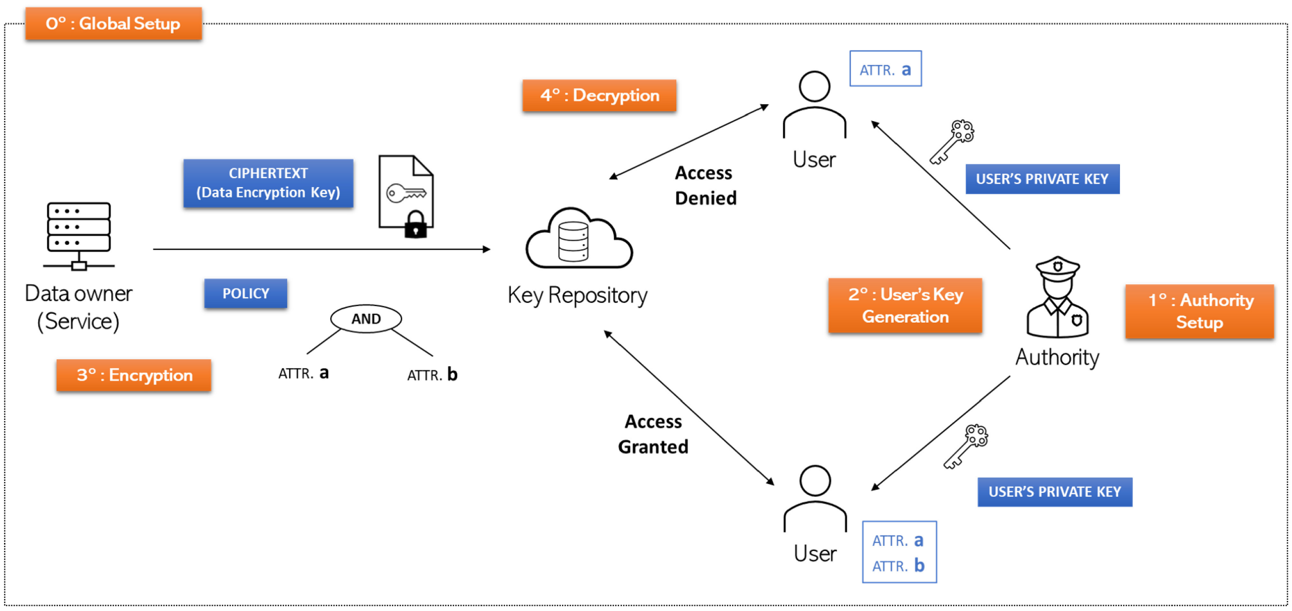


FIGURE 4. ABE scheme functions.

The successful decryption of a ciphertext depends on whether all the key shares of a user adhere to the policy linked to the ciphertext. In our application, DEKs are stored in the Key Repository of the ABE cloud service. As depicted in Figure 4, access to a particular ciphertext is granted exclusively to users who meet the specified policy criteria.

Step 4: Decrypt(CT, GP, $\{K_{i,\mathrm{GID}}\}$) $\rightarrow M$. The decryption algorithm takes in the global parameters GP, the ciphertext CT, and a collection of keys $\{K_{i,\mathrm{GID}}\}$ corresponding to pairs (attribute, identity), all with the same identity GID. It outputs the message M when the collection of attributes i satisfies the access matrix A corresponding to the ciphertext. Otherwise, decryption fails. From a mathematical viewpoint, the decryptor first obtains $H(\mathrm{GID})$. If the decryptor has the secret keys $\{K_{\rho(x),\mathrm{GID}}\}$ for a subset of rows A_x of A such that $f = (1, 0, \ldots, 0)$ is in the span of these rows, then it proceeds according to the following steps. For each such x, the decryptor computes:

$$C_{1,x} \cdot e(H(\mathrm{GID}), C_{3,x})/e(K_{\rho(x),\mathrm{GID}}, C_{2,x}) = \\ = e(g_1, g_1)^{\lambda_x} e(H(\mathrm{GID}), g_1)^{\omega_x}.$$

Then, it selects constants $c_x \in \mathbb{Z}_N | \sum_x c_x A_x = f$ and computes:

$$\prod_x (e(g_1, g_1)^{\lambda_x} e(H(\mathrm{GID}), g_1)^{\omega_x})^{c_x} = e(g_1, g_1)^s.$$

In fact, $\lambda_x = A_x \cdot v$ and $\omega_x = A_x \cdot w$, where $v \cdot f = s$ and $w \cdot f = 0$. Consequently, the message can then be obtained as $M = C_0/e(g_1, g_1)^s$.

5) SECURITY ASPECTS OF THE ABE SCHEME: MODEL DEFINITION AND FORMAL SECURITY ANALYSIS

The schema preserves security and resilience through a game played between a challenger and an attacker. The assumption here is that adversaries can only corrupt authorities statically, but key queries can be made adaptively. This slightly deviates from the standard static model, allowing an adversary to independently choose the public keys of corrupted authorities instead of having them initially generated by the challenger.

For the sake of clarity, consider the steps of the challenge, as illustrated in Figure 5. In this context, the set of authorities is represented as S, and the universe of attributes as U, with each attribute assigned to a specific authority. The security game unfolds as follows:

1) *Global Setup:* The global setup algorithm is initiated. The attacker designates a set S' of corrupt authorities out of the total set S. For non-corrupt authorities in $S - S'$, the challenger generates public-private key pairs by using the authority setup algorithm. The public keys are provided to the attacker.
2) *Key Query Phase 1:* The attacker makes key queries, submitting pairs (i, *GID*) to the challenger, where i is an attribute from a non-corrupt authority, and *GID* is an identity. The challenger responds by providing the corresponding key $K_{i,GID}$.
3) *Challenge Phase:* The attacker specifies two messages, M_0, M_1, and an access matrix (A, ρ). The matrix must satisfy constraints to ensure that the attacker cannot construct a set of keys allowing decryption in conjunction with keys obtained from corrupt authorities. The challenger encrypts M_β under access matrix (A, ρ), where β is a random coin flip ($\beta \in \{0, 1\}$).
4) *Key Query Phase 2:* The attacker may submit additional key queries (i, *GID*), adhering to constraints on the challenge matrix (A, ρ).
5) *Guess:* The attacker submits a guess β' for the coin flip β. The attacker wins if $\beta = \beta'$. The attacker's advantage in this game is defined to be $Pr[\beta = \beta'] - \frac{1}{2}$.

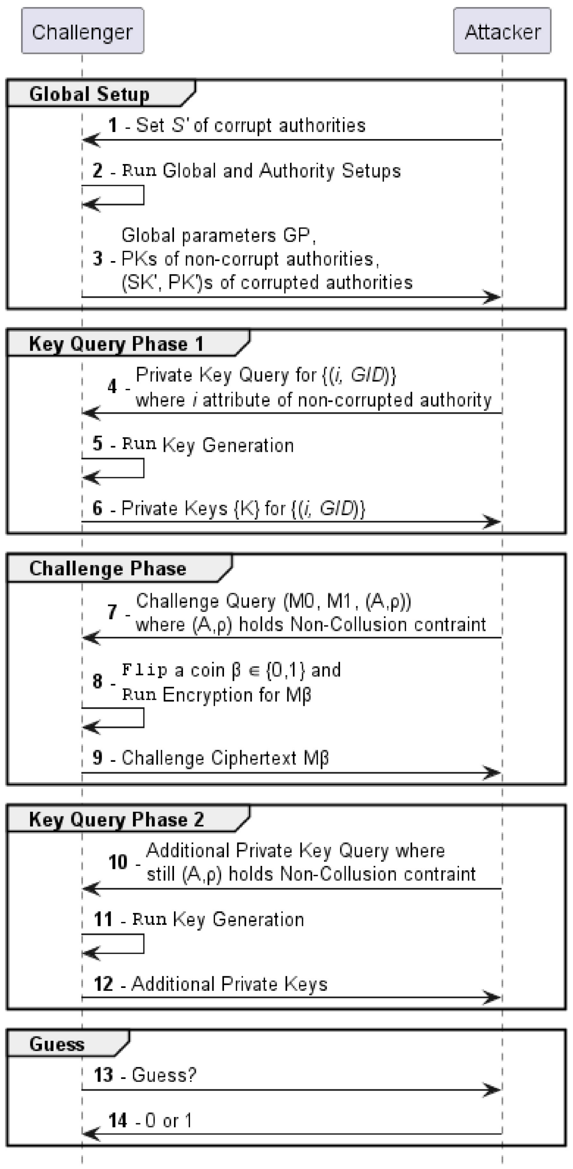

Challenger
Attacker
Global Setup
1 - Set S' of corrupt authorities
2 - Run Global and Authority Setups
3 - Global parameters GP, PKs of non-corrupt authorities, (SK', PK')s of corrupted authorities
Key Query Phase 1
4 - Private Key Query for {(i, GID)} where i attribute of non-corrupted authority
5 - Run Key Generation
6 - Private Keys {K} for {(i, GID)}
Challenge Phase
7 - Challenge Query (M0, M1, (A,ρ)) where (A,ρ) holds Non-Collusion contraint
8 - Flip a coin β ∈ {0,1} and Run Encryption for Mβ
9 - Challenge Ciphertext Mβ
Key Query Phase 2
10 - Additional Private Key Query where still (A,ρ) holds Non-Collusion contraint
11 - Run Key Generation
12 - Additional Private Keys
Guess
13 - Guess?
14 - 0 or 1

FIGURE 5. **Security game sequence diagram.**

A Multi-Authority CP-ABE system is secure (against static corruption of authorities) if all polynomial time attackers have at most a negligible advantage in this security game.

In order to prevent collusion, the main strategy consists of using a hash function on the user's global identity. The output of this function $H(\text{GID})$ is the element of a bilinear group (see Appendix A) that ties keys together. Based on the theory of the linear secret-sharing schemes (introduced in Appendix B), the key idea is to handle the decryption process on the nodes x of the access tree, such that it is possible for a user to recover a group element of the form $e(g, g)^{\lambda_x} \cdot e(g, H(\text{GID}))^{w_x}$. The first element of this group element is a share λ_x of the secret s to recover. Thus, these shares need to be combined for deciphering the message. However, they are masked by a share w_x of the 0 element in the exponent of the second factor, that is the one with base $e(g, H(\text{GID}))$. The algorithm in [46] allows simultaneously reconstructing the secret s and unblinding it. In fact, if a user identified by GID satisfies the access tree, he can obtain s by reconstructing it in the exponent by raising group elements to the proper base. At the same time, this operation will also reconstruct the shares of 0 and thus, in the end, the terms $e(g, H(\text{GID}))$ will disappear. In particular, the encryption algorithm ciphers the message M with $e(g_1, g_1)^s$, where g_1 is a generator of the subgroup G_{p_1}, and s is a randomly selected value in $\mathbb{Z}_N$, with $N = p_1p_2p_3$. The value s is then split into shares λ_x through the A matrix, and the value 0 is split into shares w_x. Hence, in order to recover the information message M, the decryption algorithm has to recover the blinding factor $e(g_1, g_1)^s$, so introducing terms in the form $e(g_1, H(\text{GID}))^{w_x}$. If the decryptor has a satisfying set of key s with the same identity GID, these additional terms will cancel from the final result, since the w_x elements are shares of 0. Instead, if two users with identities GID and GID' attempt to collude and combine their keys, then there are two types of terms, those of the form $e(g_1, H(\text{GID}))^{w_x}$ and others of the form $e(g_1, H(\text{GID}'))^{w_{x'}}$. These different types of terms do not cancel with each other, thereby preventing to recover $e(g_1, g_1)^s$ and thus to decipher the message M.

The formal proof of security of this model against collusion attacks, making use of the dual system encryption methodology, is quite lengthy, and it is given in [46, Secs. IV-A and 4.2, Appendix C]. Here, we provide an outline of the strategy adopted, inviting the interested reader to refer to the original document for all the details, as well as to [65]. By employing a form of the dual system encryption technique, the authors of [46] address the challenges specific to the multi-authority setting. In this framework, keys and ciphertexts can be either normal or semi-functional, with distinct decryption capabilities. The security proof involves a series of games employing a hybrid argument, where the challenge ciphertext is first transformed into a semi-functional form, followed by a gradual transformation of the keys to a semi-functional state. The key challenge lies in ensuring indistinguishability between these games, preventing the simulator from discerning the transformation of keys from normal to semi-functional during the testing process. To tackle this, the authors adopt an approach involving nominal semi-functionality, where both key and ciphertext have semi-functional components that cancel out upon decryption. Moreover, in the presence of multiple authorities lacking coordination, the authors introduce innovative solutions, such as the use of temporary "blinding factors" to conceal nominal semi-functionality. These factors, active for one key at a time, are strategically switched between two subgroups in the multi-authority case, preserving semi-functionality while averting information leakage about the subgroup in which the semi-functional components operate.

In conclusion, this section highlights our dual objective, consisting in both unified management system for orchestration services and a robust security solution addressing confidentiality and compliance concerns. To achieve the

TABLE 2. Comparison with other Kubernetes cloud bursting solutions.

Solution	Vendor	Supported public cloud providers	Approach	Security		
				Access control	Network security	Data privacy
KIP [39]	Elotl	EKS, GKE	Virtual Kubelet	RBAC	Dedicated interconnect or VPN	N/A
KubeSlice [57]	Avesha	EKS, GKE, AKS	Virtual slice across multiple clusters	RBAC	Secure VPN tunnels + zero trust	N/A
ACK One [67]	Alibaba Cloud	ACK (mandatory) + many others	Centralized management via registration to ACK portal	RBAC	TLS-based encryption	N/A
CloudBees Build Acceleration [10]	CloudBees	EKS, GKE, AKS	Cluster Manager installed anywhere + Electric Agents running in worker nodes	RBAC	TLS encryption between agents	N/A
Anthos [7]	Google	GKE, EKS	Configuration synchronization via GitOps	RBAC	TLS-based service mesh	N/A
Proposed solution	Open-source	Multiple	Label-based operations for both orchestration and security	ABAC + ABE	TLS-based encryption	ABE

latter, we advocate for employing CP-ABE in a hybrid mode, providing fine-grained access control, adaptable policies, and an optimized balance between security and efficiency. This approach, as depicted in Figure 3, enables authorized users to access encrypted data stored within service volumes, streamlining data retrieval.

We have explored the correlation between attribute values, label values, and encryption policies, highlighting the critical role of congruence for streamlined configuration and management. This unification enhances transparency and user experience by eliminating redundant configurations and harmonization processes.

Furthermore, the discussion on the expressive power of ABE policies, encompassing logical expressions, underlines the versatility of ABE schemes and their potential to accommodate a broad spectrum of access control scenarios.

Finally, we describe the ABE scheme used in our research, motivated by its features, and analyzed its security. Importantly, we emphasize again that the selection of this specific scheme is not a rigid constraint. This flexibility reflects the dynamic nature of the security solutions, allowing for adaptation to different requirements. For these reasons, we believe that these concepts can improve security, efficiency, and versatility in the considered framework.

C. COMPARISON WITH OTHER CLOUD BURSTING APPROACHES

In this section, our proposal is compared with available counterparts presented in Section II. A schematic comparison is reported in Table 2. This table summarizes the main features of cloud bursting solutions of interest for our purposes, i.e., support by public cloud vendors, strategic approach, and security features. Whilst different solutions follow different approaches, all of them seem suitable to offload computing workload in public clouds. However, for what concerns security, it is handled separately from orchestration functions. In fact, whereas communications security is typically handled via TLS-based encryption, access control is enforced only via RBAC, which suffers from known limitations. Finally, none of the considered solutions natively supports strong data encryption, which is a significant shortcoming. Although it is possible to add encryption modules for handling data security as additional services in virtual clusters running on public resources, they have to be managed manually by developers. This increases the complexity of service management when cloud bursting is necessary. Instead, by jointly carrying out orchestration, including bursting, and data protection through labeling, which is supported natively by K8s, this feature comes for free, without any additional burden for developers.

V. VALIDATION

In this section, we present the findings from our proof of concept (POC) for the proposed attribute-based management model for secure K8s cloud bursting. The main research question addressed by this POC is: Can attributes be used to establish a centralized system capable of simultaneously providing data confidentiality and facilitating efficient cloud bursting?

We evaluated the feasibility and effectiveness of this attribute-based management model through a specific proof of concept. To showcase the achievable performance, we produced a POC available on our GitHub repository [54]. This POC offers a straightforward scenario for assessing our proposal.

A. PROOF OF CONCEPT SET UP

The experimental setup included two interconnected clouds. It emulates a system either where a service is hosted on a local cloud only or where a local cloud connects to a public cloud through a dedicated virtual connection service, such as Azure ExpressRoute. The ABE service, crucial to our

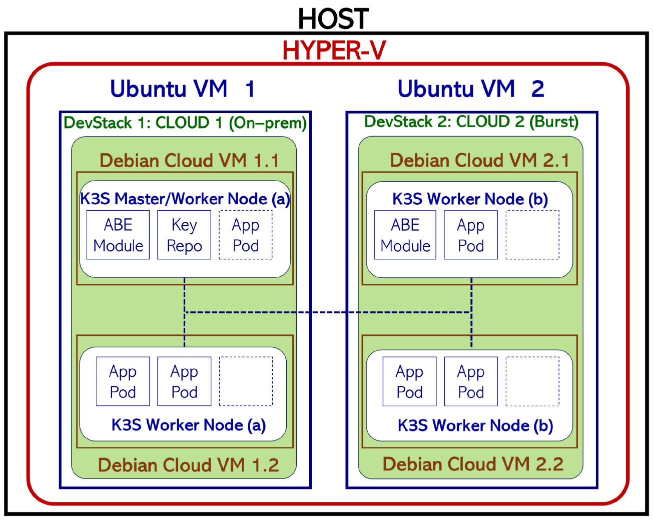
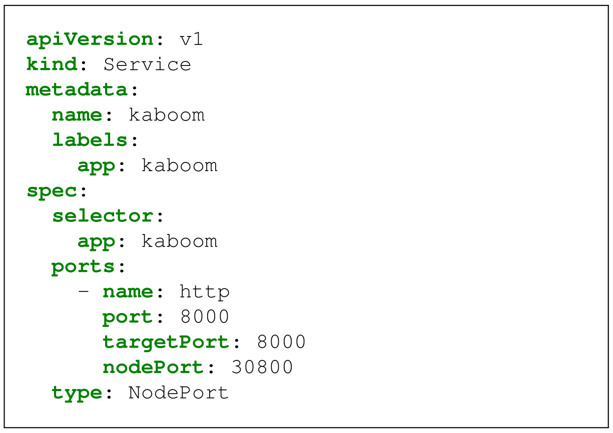


FIGURE 6. Test-bed components.



```
apiVersion: v1
kind: Service
metadata:
  name: kaboom
  labels:
    app: kaboom
spec:
  selector:
    app: kaboom
  ports:
    - name: http
      port: 8000
      targetPort: 8000
      nodePort: 30800
  type: NodePort
```

FIGURE 7. Yaml of Kubernetes Service.

approach, is hosted within this cluster. It serves to encrypt, decrypt, and store data of string type, illustrating the core functions of our model.

The POC environment was hosted on a personal computer featuring an Intel i7-8700 CPU, 32GB RAM, and a 1TB SSD. The test environment is illustrated in Figure 6. It encompassed two distinct Ubuntu 22.04 LTS virtual machines (VMs), each one running a DevStack instance, thus effectively creating two separate cloud deployments (On-prem and Burst cloud). Each VM was allocated 6 virtual cores (vCPU), 12GB RAM, and 64GB SSD, ensuring a balanced testing environment. In turn, each DevStack cloud instance ran two Debian Cloud VMs, each one equipped with 2 vCPUs, 4GB RAM, and 20GB SSD. Each Debian VM ran a Kubernetes instance using K3s multi-cloud, one acting as both cluster master and worker node, and the other three as worker nodes. Thus, all Kubernetes instances were able to run application pods. Figure 6 shows the label (in brackets) of all Kubernetes nodes. In more detail, in order to configure the *on-premise/burst* attribute on the Kubernetes nodes, we labeled each node using `kubectl` by the label `a` for the on-premise nodes inside the first cloud and by the label `b` for the burst nodes running inside the second cloud, respectively. The commands used for labeling nodes as *on-premise* and *burst* are as follows:

```
kubectl label nodes <nn> node-type=a
where nn="master1a worker1a"

kubectl label nodes <nn> node-type=b
where nn="worker2b worker2b1"
```

These labels are then used in the `affinity` section of the Kubernetes Deployment manifest to automatically guide scheduling of pods.

We produced some YAML manifest files to configure the Kubernetes orchestration in the proposed scenario.

The first manifest file, shown in Figure 7, defines a Kubernetes Service of type NodePort. The Service is named `kaboom` and is labeled with `app: kaboom`. Its purpose is to expose the application to the external. In particular, this Service exposes port 8000 and maps it to the node port 30800 and targets Pods labeled `app: kaboom`. When a user's request accesses the node on port 30800, the Service forwards the traffic to one of the pods listening on port 8000.

The second manifest file, shown in Figure 8, defines a Kubernetes Deployment. It creates a pod labeled with `app: kaboom`, runs the container `mrcolorrain/kaboom:latest`, and exposes port 8000 on the container.

The `Readiness Probe` is used to check the health of the application. The `resources` section sets resource requests for the pod. The `affinity` section specifies that the pod is expected to be scheduled on the nodes labeled with `node-type: a`, i.e., on-premise nodes. The strategy is set to `Rolling Update`, which means that the old pods are gradually replaced with new pods. This strategy ensures that there is no downtime during the update process and that the application remains available to users.

The last manifest file, shown in Figure 9, defines an autoscaler for the `kaboom` deployment. The autoscaler targets the `kaboom` deployment and scales between a minimum of 1 and a maximum of 40 replicas. It scales based on CPU utilization, targeting an average of 50%. The scale-down stabilization window is set to 90 seconds. Note that the horizontal pod autoscaler targets a maximum of 40 replicas for the `kaboom` deployment. This is due to an important aspect of resource utilization: considering that each `kaboom` pod consumes 0.128 CPU, a maximum of 40 replicas would require up to 5.12 CPUs in total. Given that the two VMs hosting the on-prem worker nodes are configured with only 2 vCPUs each, and one of them also runs the Kubernetes master, any deployment beyond 32 replicas would certainly trigger cloud bursting to meet the resource requirements.

By using the manifest files shown, we created a test computing environment where the `kaboom` application is exposed to the external world, automatically scales based on

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kaboom
  labels:
    app: kaboom
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kaboom
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
  revisionHistoryLimit: 1
  template:
    metadata:
      labels:
        app: kaboom
    spec:
      containers:
        - name: kaboom
          image: mrcolorrain/kaboom:latest
          ports:
            - containerPort: 8000
          readinessProbe:
            httpGet:
              path: /
              port: 8000
            initialDelaySeconds: 5
            periodSeconds: 5
          resources:
            requests:
              cpu: 128m
              memory: 128Mi
      affinity:
        nodeAffinity:
          preferredDuringScheduling
           ↪ IgnoredDuringExecution:
          - weight: 100
            preference:
              matchExpressions:
              - key: node-type
                operator: In
                values:
                - a
```

FIGURE 8. Yaml of Kubernetes Deployment.

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: kaboom
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: kaboom
  minReplicas: 1
  maxReplicas: 40
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 90
```

FIGURE 9. Yaml of Autoscaler.

CPU utilization, and is preferably scheduled on nodes *on-premise* labeled as a. This is actually the cloud architecture characterizing the POC scenario illustrated above, enhanced by the integration of the Kubernetes management functions with ABE for secure cloud bursting.

This implementation can be regarded as the starting point for additional research in a more complex situations, such as strengthening security of multi-tenant Kubernetes environments loaded with multiple applications. In these challenging situations, it is critical to achieve a fine control of application deployment which is compliant with policies and regulations. Our approach, based on node affinity and pod anti-affinity rules combined with ABE policies, can help to achieve this control. In fact, it allows protecting access to data and isolating target applications from both specific nodes and other services. For instance, if certain applications should be prevented from being scheduled on cloud nodes or near another specific application, it is possible to use the manifest files shown as blueprint to include a node affinity rule and a pod anti-affinity rule for comprehensive scheduling control, as shown in Figure 10.

In this example, the node affinity rule confines the `kaboom` application to on-premises nodes, while the pod anti-affinity rule guarantees that `kaboom` pods do not share an environment with `anti-kaboom` pods. This dual-layered approach offers robust isolation and fine-grained control over Kubernetes deployments.

B. ANALYSIS OF ABE FUNCTIONS

The ABE critical components were developed by using the Rust programming language. Specifically, the ABE module was crafted by using the Rocket framework version 5.0 [21]. Cryptographic primitives integral to the AW11 scheme implementation are based on the Rabe library version 0.2.7 [4]. We emulated user interactions through both the Web interface and the API calls issued through a terminal. The ABE components run as Kubernetes pods, as shown in Figure 6. The ABE module can be accessed either through the Web interface or the terminal, and it presents a menu comprising the following options:

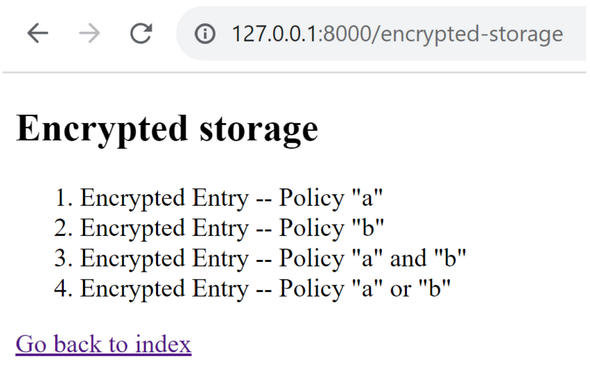
1) Show encrypted storage
2) Decrypt storage
3) Update storage

1) OPTION 1: SHOW ENCRYPTED STORAGE

The first option, accessible to anyone, allows accessing and viewing the encrypted storage. As shown in Figure 11,

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kaboom
spec:
  replicas: 3
  selector:
    matchLabels:
      app: kaboom
  template:
    metadata:
      labels:
        app: kaboom
    spec:
      affinity:
        nodeAffinity:
            requiredDuringScheduling
             ↪ IgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: node-type
                operator: NotIn
                values:
                - b
        podAntiAffinity:
          requiredDuringScheduling
           ↪ IgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - anti-kaboom
              topologyKey:
               ↪ "kubernetes.io/hostname"
```

FIGURE 10. Example showing the use of affinity and anti-affinity to change cloud bursting and apps deployment behavior.


127.0.0.1:8000/encrypted-storage

Encrypted storage

1. Encrypted Entry -- Policy "a"
2. Encrypted Entry -- Policy "b"
3. Encrypted Entry -- Policy "a" and "b"
4. Encrypted Entry -- Policy "a" or "b"

Go back to index

FIGURE 11. Example of encrypted storage (screenshot of the Web interface).

which presents a screenshot of the Web interface, this list reveals the number of encrypted entries within the ABE volume. Additionally, it provides an instructive glimpse into the associated policies for each entry.

In Figure 11, the example showcases the dynamic interplay of policies. The first two entries are linked to simple policies encompassing a single system attribute. In contrast, the subsequent entries are tied to slightly more intricate policies that incorporate not only attributes but also logical operators.

```json
{
    "_gid": "bob",
    "_attr": [
        [
            "b",
            {
                "x": [
                    5712868850552490035,
                    3340114564258679341,
                    7948731980494642711,
                    194928839779717989
                ],
                "y": [
                    11833180424976684086,
                    17503833986257901125,
                    16512980884993783250,
                    2315106061390818652
                ],
                "z": [
                    13701724635261595198,
                    7942669771980227583,
                    14101786126588513581,
                    3102591607448728316
                ]
            }
        ]
    ]
}
```

FIGURE 12. Example of user key.

This visual demonstration underscores the ABE potential to extend policy complexity, enhancing the expressive capabilities of the protection system.

2) OPTION 2: DECRYPT STORAGE

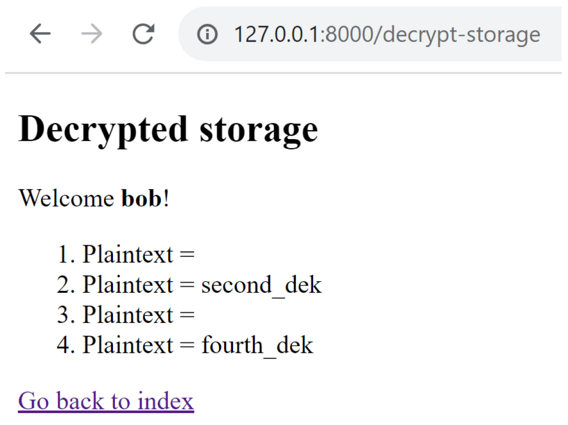
The second option, labeled as *Decrypt Storage*, requires the introduction of a user key before the decryption process can proceed.

In Figure 12, we can observe an example of a user key utilized in our tests. This private key is formatted in JSON, containing references to the key owner, in this case, bob, and the attributes associated with it. In this instance, it holds a single attribute b, defined with specific coordinates. This key functions as a passkey, instructing the system on what content to decrypt and what to keep concealed.

Once the user key has been entered, the screen depicted in Figure 13 is displayed. In accordance with the policies outlined in Figure 11, the user possession of the b attribute grants him access solely to the second and fourth entries. Consequently, the user can retrieve two DEKs (see also Figure 3), providing him with the means to securely interact with data services within the cluster.

3) OPTION 3: UPDATE STORAGE

The third option concludes the menu by enabling the addition of a new entry to the encrypted storage. This procedure involves supplying two crucial pieces of information: the plaintext content and the policy used for encryption. Once


127.0.0.1:8000/decrypt-storage

Decrypted storage

Welcome **bob**!

1. Plaintext =
2. Plaintext = second_dek
3. Plaintext =
4. Plaintext = fourth_dek

Go back to index

FIGURE 13. Example of decrypted storage (screenshot of the Web interface).

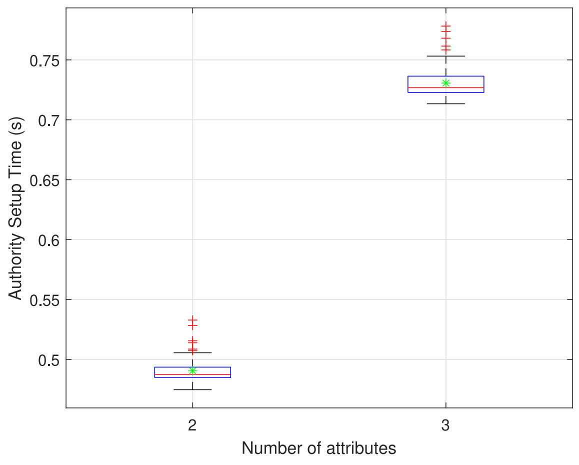


FIGURE 14. Box plot of the authority setup time as a function of the number of attributes. The green asterisk indicates the average value.

TABLE 3. Performance of ABE-related functions.

		#2 attributes	#3 attributes
Authority Setup	Average	0.4906	0.7307
	SD	0.0096	0.0122
	Min	0.4748	0.7134
	Max	0.5329	0.7782
User Private Key Generation	Average	0.0594	0.0885
	SD	0.0016	0.0019
	Min	0.057	0.0855
	Max	0.0656	0.0974
Encryption	Average	1.2276	1.6089
	SD	0.1225	0.118
	Min	1.1779	1.5565
	Max	2.1592	2.4436
Decryption	Average	0.55	0.554
	SD	0.0073	0.0133
	Min	0.5431	0.5443
	Max	0.5838	0.6454

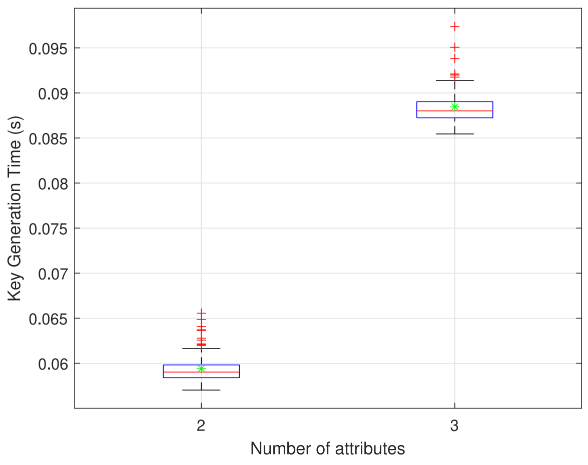


FIGURE 15. Box plot of the user private key generation time as a function of the number of attributes. The green asterisk indicates the average value.

these details are provided, the system undertakes the encryption process automatically, followed by the seamless storage of the encrypted entry within the ABE volume.

C. ABE-FUNCTIONS PERFORMANCE

This section presents the performance results obtained by using the ABE scheme. Experiments have been carried out on one of the Ubuntu VMs used in the POC (see Section V-A). We used the Rabe library, which is a Rust-based library implementing ABE. We selected the AW11 scheme [4], [46], as mentioned above. The encryption process converts 256-bit plaintext segments, managed as a symmetric key, according to Section IV-B. We collected the performance metrics related to the four main functions of an ABE scheme: authority setup, user private key generation, encryption, and decryption. All of them are sketched in Figure 4.

In performance evaluation, We varied the number of attributes used in the policy and the user's secret key. In particular, we used 2 and 3 attributes. The experiment was repeated 100 times for each attribute count. Numerical results are reported in Table 3, in terms of average latency, its standard deviation, minimum and maximum values. All results are expressed in seconds. In addition, for each of these functions, we included a box plot diagram, in order to better appreciate their statistical properties. These diagrams also show the average value, marked by a green asterisk. All latency values are grouped in a quite compact range (interquartile range), showing only a few significant outliers for encryption (Figure 16) and decryption (Figure 17) functions. Differently, authority setup times (Figure 14) and key generation times (Figure 15) show more grouped values.

Observing Table 3 and box plots diagrams, it can be observed a generalized increase of the processing time when switching from 2 to 3 attributes, which indicates an increase in complexity. Notably, the increase with the number of attributes in the authority setup time (+48.9% on average), user private key generation (+48.9% on average), and encryption (+31% on average) functions is more pronounced compared to the decryption function, where it is less steep (+0.66% only on average).

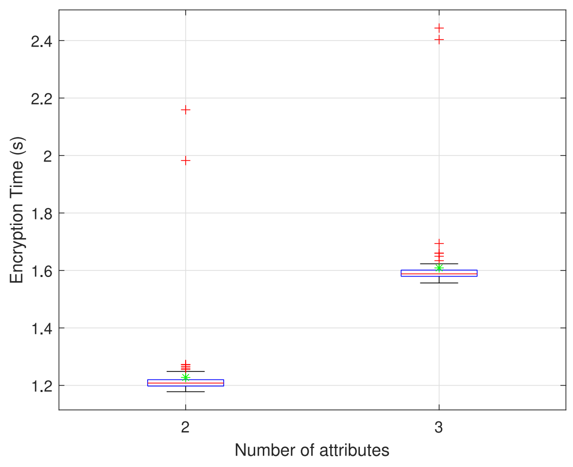


FIGURE 16. Box plot of the encryption time as a function of the number of attributes. The green asterisk indicates the average value.

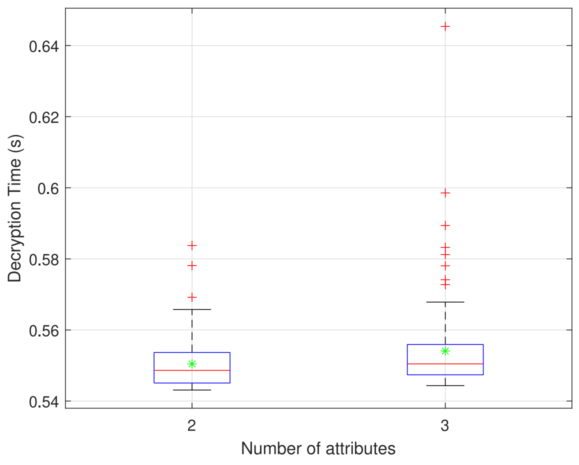


FIGURE 17. Box plot of the decryption time as a function of the number of attributes. The green asterisk indicates the average value.

In order to evaluate the average payoff due to the presence of ABE, we evaluated the added latency per user session, depicted in Figure 3 (service data access). In that figure it appears that, in general, only the decryption function is invoked for completing a new session by the user, whereas the other functions are used only occasionally (setup and encryption of the DEK). Thus, in order to evaluate the average added latency, we measured the added contribution due to the decryption function as well as the protocol used for interacting with the ABE module, which is accessible through a REST API. Figure 18 shows these results, identifying the two above delay contribution: DEK decryption and REST interaction. The overall added latency due to our solution is slightly less than 0.6 seconds, with a negligible 95% confidence intervals. This latency is essentially due to the DEK decryption, and does not depend in a significant way from the number of attributes used to implement the policy. It seems to be a reasonable price to obtain an improved security.

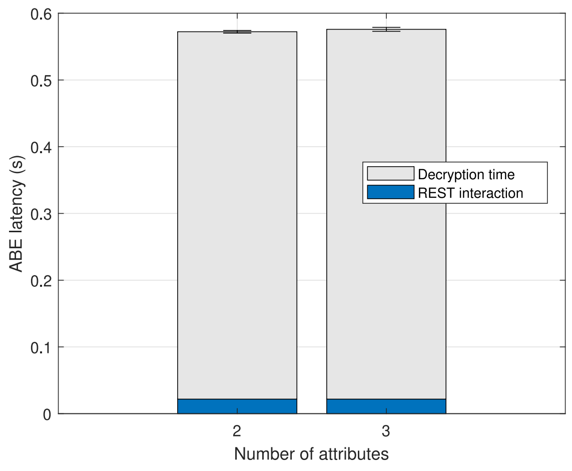


FIGURE 18. Latency associated with ABE function with the relevant 95% confidence intervals. Relative weights of decryption function and REST calls contributions are highlighted.

Although we explored some key use-cases, an exhaustive exploration of attribute usage is beyond the scope of this paper. Mosteiro-Sanchez et al. [51] have already conducted comprehensive experiments in this regard. Our emphasis lies in the broader applicability of the insights obtained. While we specifically implemented the AW11 scheme, [51] serves not only to deepen the performance understanding of AW11, but is a rather valuable reference also for assessing the performance of different ABE libraries.

D. DISCUSSION

We validated through a POC the system's ability to adjust cloud resources based on computational requirements using the cluster setup described in previous blueprints, which is critical in cloud bursting scenarios.

The ABE module revealed its capabilities through the menu options. *Show Encrypted Storage* illustrated the potential of the module to manage intricate access policies, *Decrypt Storage* emphasized secure data access through user keys, and *Update Storage* showcased the seamless integration of encryption processes. The ABE user-centric module design promotes secure data interaction. It aligns with modern data security paradigms, maintaining data privacy even during burst operations. The visual policy representation empowers administrators with effective access control.

In summary, the usage of labels common to Kubernetes configuration (nodes and app pods) and ABE policies allows *jointly* configure cloud bursting and security features, leveraging the potential of the attribute-based approach.

In terms of implications and future prospects, our experiments validate the viability of our attribute-based management model for secure cloud bursting. The convergence of Kubernetes and ABE presents a promising solution for organizations seeking secure and scalable cloud deployments. Future work can address scalability challenges and explore advanced security layers.

VI. CONCLUSION

In this paper, we introduce a robust orchestration scheme tailored for secure cloud bursting. This scheme addresses the complexities, cost challenges, and stringent data protection compliance requirements by harnessing the combined capabilities of K8s and ABE.

By incorporating ABE, our approach achieves granular encryption and provides cloud resources with suitable confidentiality. At the same time, cloud bursting empowers the extension of computational tasks beyond the scope of a local primary cloud environment. The synergy of these two technologies establishes a cohesive management framework, guaranteeing secure access to bursting services and streamlined deployment of excess workloads to the cloud, all facilitated by Kubernetes.

Significantly, our contributions encompass the design blueprint for an attribute-based cloud bursting authorization system within Kubernetes. Through the application of ABE, we enhance data security to adhere to regulatory mandates and mitigate data-related apprehensions. By bridging this critical gap, our approach stands as a holistic solution that aligns with both security and privacy regulations, meeting the contemporary requisites of cloud-driven systems.

Practical feasibility of our proposal is demonstrated by the implementation of a proof of concept, that shows its potential to be seamlessly integrated into real-world scenarios.

Future research will consider the integration of the proposed solution with artificial intelligence algorithms to proactively infer the need of resorting to bursting operations. In fact, purely relying on a reactive approach the latency of the process can either lead to temporary violation of service level agreements - loose approach - or allocation of excessive resources - conservative approach.

APPENDIX

In these appendices, we report some useful definitions, some notions about bilinear groups as well as the very basics of linear secret-sharing schemes. These concepts are used in the paper of Lewko and Waters [46], and are instrumental for a better understanding of the mathematics about ABE used in this manuscript.

A. BILINEAR GROUPS

Bilinear groups of composite order are groups with an efficient bilinear map where the group order is a product of two large primes. Such groups are constructed from pairing friendly curves over a finite field [25], [26]. The following assumptions are formulated for a bilinear group G of order N, where $N = p_1p_2p_3$ is the product of 3 different primes. Now, let $e : G \times G \rightarrow G_T$ denote a bilinear map, that is a function with the following properties:

1) $\forall g, h \in G,\ a, b \in \mathbb{Z}_N,\ e(g^a, h^b) = e(g, h)^{ab}$ (bilinear),
2) $\exists g \in G | e(g, g)$ has order n in G_T (non-degenerate).

A group generator $\mathcal{G}$ is an algorithm which takes a security parameter λ as input and produces a description of a bilinear group G, i.e., $(p_1, p_2, p_3, G, G_T, e)$, where both G and G_T are cyclic groups of order N, and e is a bilinear map. It is assumed that the group operations in G and G_T and the map e can be computed in polynomial time with respect to λ, and that the group descriptions include generators of the respective cyclic groups, according to the above definition. Let G_{p_1} denote the subgroup of order p_1 in G. If $g_i \in G_{p_i}$ and $g_j \in G_{p_j}$ for $i \neq j$, then $e(g_i, g_j) = 1$. The construct $g_1 \overset{R}{\leftarrow} G_{p_1}$, indicates that g_1 is a random generator of G_{p_1}, Thus, it is not the identity element. The interested reader can found additional details on composite order bilinear group in [46, Appendix A] and references therein.

According to the class of General Subgroup Decision Assumptions described in [19], in a bilinear group G of order $N = p_1p_2 \dots p_n$, a subgroup of order $\prod_{i \in S} p_i$ exists for each subset $S \subseteq \{1, \dots, n\}$. Let S_0, S_1 denote two distinct subsets, assuming that it is hard to distinguish a random element from the subgroup associated with S_0 from another of the subgroup associated with S_1. This is true even though it is possible to have random elements from subgroups associated with several subsets Z_i, which satisfy one of the following two mutually exclusive conditions: $S_0 \cap Z_i = S_1 \cap Z_i = \emptyset$, or $S_0 \cap Z_i \neq \emptyset \neq S_1 \cap Z_i$.

In [46], the entire system is confined to the subgroup G_{p_1} in G except $H(\cdot)$, which instead maps identities onto random group elements in G. The subgroups G_{p_2} and G_{p_3} are used in the security proof, which makes use of the dual system encryption technique, where keys and ciphertexts can be either normal or semi-functional. In the approach proposed in [46] and used in this work, normal keys and ciphertexts are contained in the subgroup G_{p_1}, while semifunctional keys and ciphertexts use elements of G_{p_2} and G_{p_3}. Thus, G_{p_2} and G_{p_3} form the semi-functional space orthogonal to G_{p_1}.

B. LINEAR SECRET-SHARING SCHEME

Given a set of parties $\mathcal{P} = \{P_1, \dots, P_n\}$, a collection is defined as $\mathbb{A} \subseteq 2^{\{P_1,\dots,P_n\}}$. An access structure is a collection $\mathbb{A}$ of non-empty subset, i.e., $\mathbb{A} \subseteq 2^{\{P_1,\dots,P_n\}} - \emptyset$. The sets in $\mathbb{A}$ are called authorized sets, those outside $\mathbb{A}$ are called unauthorized sets.

A Linear Secret-Sharing Scheme (LSSS) [52] can be defined as follows. Given a set of parties $\mathcal{P}$, a secret sharing scheme Λ is defined *linear* over $\mathcal{P}$ (over $\mathbb{Z}_p$) if

- The shares of each party form a vector in $\mathbb{Z}_p$.
- An $l \times n$ matrix A exists with the role of share-generating matrix for Λ. Given a function $\rho(x) : x \in \{1, .., l\} \rightarrow \mathcal{P}$, each row x of A is labeled by it. If the secret to be shared is $s \in \mathbb{Z}_p$ and $r_2, \dots, r_n \in \mathbb{Z}_p$ are random values, then it is possible to obtain the column vector $v = (s, r_2, \dots, r_n)$ such that the product Av is the vector of l shares of the secret s according to Λ. The x-th share of Av, denoted as $(Av)_x = A_x v$, belongs to party $\rho(x)$.

From this definition, the linear reconstruction property follows. Under the assumption that Λ is an LSSS for $\mathbb{A}$, if S

is an authorized set, it is possible to define $I = \{x|\rho(x) \in S\}$, with $I \subseteq \{1, \dots, l\}$. Thus, the vector $f = (1, 0, \dots, 0)$ is in the set of rows of A indexed by I. Furthermore, a set of constants $\{\omega_x \in \mathbb{Z}_p\}_{x \in I}$ exists such that $s = \sum_{x \in I} \omega_x \lambda_x$ for any valid shares $\{\lambda\}_x$ of the secret s according to Λ. The above constants $\{\omega_x \in \mathbb{Z}_p\}$ can be found in polynomial time with respect to the size of A. This also means that $\lambda_x = A_x \cdot v$ where $v \cdot f = s$.

For the construction of the composite order group, given an LSSS matrix A over $\mathbb{Z}_N$, where $N = p_1 p_2 p_3$ is the product of 3 different primes, S is an authorized set if the rows of A that are labeled by the elements in S through the function ρ have the vector f as defined above in their span modulo N. The access policies can be described through monotonic boolean formulas and there are standard techniques to derive a LSSS matrix from any monotonic boolean formula. The boolean formula can be represented as an access tree, where interior nodes consists of AND and OR gates, whereas the leaves correspond to attributes. Clearly, the number of rows in a LSSS matrix is equal to the number of leaves in the access tree. Additional details can be found in [46, Appendix G].

ACKNOWLEDGMENT

The authors acknowledge Università degli Studi di Perugia and MUR for support within the projects VITALITY and RESTART.

REFERENCES

[1] (Amazon Web Serv., Inc., Seattle, WA, USA). *Amazon Elastic Kubernetes Service (Amazon EKS)*. Accessed: Jun. 8, 2023. [Online]. Available: https://aws.amazon.com/eks/?nc1=h_ls

[2] (google, Mountain View, CA, USA). *Google Kubernetes Engine*. Accessed: Jun. 8, 2023. [Online]. Available: https://cloud.google.com/kubernetes-engine

[3] (Int. Bus. Mach. Corp., Armonk, NY, USA). *Ibm Kubernetes Service*. Accessed: Jun. 8, 2023. [Online]. Available: https://www.ibm.com/cloud/kubernetes-service

[4] "Module aw11 rabe." Accessed: Jan. 5, 2024. [Online]. Available: https://docs.rs/rabe/latest/rabe/schemes/aw11/index.html

[5] (Oracle Comput. Softw. Co., Austin, TX, USA). *Oracle Cloud Native Services—Container Engine for Kubernetes*. Accessed: June 8, 2023. [Online]. Available: https://www.oracle.com/cloud/cloud-native/container-engine-kubernetes/

[6] "Security best practices for Kubernetes deployment." Mar. 2019. [Online]. Available: https://kubernetes.io/blog/2016/08/security-best-practices-kubernetes-deployment/

[7] "Anthos." Dec. 2023. [Online]. Available: https://cloud.google.com/anthos

[8] (Palo Alto Networks, Inc., Santa Clara, CA, USA). *Cloud-Native Applications Protection Platform*. Dec. 2023. [Online]. Available: https://www.paloaltonetworks.com/prisma/cloud/cloud-native-application-protection-platform

[9] "Configuration best practices: Using labels." Jul. 2023. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/overview/#using-labels

[10] (CloudBees Softw. Co., San Jose, CA, USA). *Configuring Cloudbees Build Acceleration for Agent Cloud Bursting*. (Dec. 2023). [Online]. Available: https://docs.cloudbees.com/docs/cloudbees-build-acceleration/latest/configuration-guide/config-accelerator-agents-for-cloud-burst

[11] "Kubernetes autoscaler." github. Dec. 2023. [Online]. Available: https://github.com/kubernetes/autoscaler

[12] (Microsoft Corp. Technol. Corp., Redmond, WA, USA). *Securing Kubernetes Workloads In Hybrid Settings With Aporeto*. (Dec. 2023). [Online]. Available: https://cloudblogs.microsoft.com/opensource/2018/08/31/securing-kubernetes-workloads-hybrid-cloud-aporeto/

[13] (Amazon Web Serv., Inc., Seattle, WA, USA). *TLS-Enabled Kubernetes Clusters With ACM Private CA and Amazon EKS*, (Dec. 2023). [Online]. Available: https://aws.amazon.com/it/blogs/security/tls-enabled-kubernetes-clusters-with-acm-private-ca-and-amazon-eks-2/

[14] "Virtual kubelet." Dec. 2023. [Online]. Available: https://virtual-kubelet.io/

[15] R. Ahuja and S. K. Mohanty, "A scalable attribute-based access control scheme with flexible delegation cum sharing of access privileges for cloud storage," *IEEE Trans. Cloud Comput.*, vol. 8, no. 1, pp. 32–44, Mar. 2020.

[16] S. Ameer, J. Benson, and R. Sandhu, "An attribute-based approach toward a secured smart-home IoT access control and a comparison with a role-based approach," *Information*, vol. 13, no. 2, p. 60, 2022.

[17] D. Balouek-Thomert, E. G. Renart, A. R. Zamani, A. Simonet, and M. Parashar, "Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows," *Int. J. High Perform. Comput. Appl.*, vol. 33, no. 6, pp. 1159–1174, 2019.

[18] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi, "A unified model for the mobile-edge-cloud continuum," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–21, Apr. 2019.

[19] M. Bellare, B. Waters, and S. Yilek, "Identity-based encryption secure against selective opening attack," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2010/159, 2010.

[20] P. Benedetti, M. Femminella, G. Reali, and K. Steenhaut, "Reinforcement learning applicability for resource-based auto-scaling in serverless edge applications," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Other Affil. Events (PerCom Workshops)*, 2022, pp. 674–679.

[21] S. Benitez. "Meet rocket." Accessed: Jan. 5, 2024. [Online]. Available: https://rocket.rs/

[22] S. Bera, S. Prasad, Y. Sreenivasa Rao, A. K. Das, and Y. Park, "Designing attribute-based verifiable data storage and retrieval scheme in cloud computing environment," *J. Inf. Secur. Appl.*, vol. 75, Jun. 2023, Art. no. 103482.

[23] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Security Privacy (SP'07)*, 2007, pp. 321–334.

[24] S. Böhm and G. Wirtz, "Cloud-edge orchestration for smart cities: A review of Kubernetes -based orchestration architectures," *EAI Endorsed Trans. Smart Cities*, vol. 6, no. 18, p. e2, May 2022.

[25] D. Boneh, "Bilinear groups of composite order," in *Proc. 1st Int. Conf. Pair.-Based Cryptogr.*, 2007, p. 1.

[26] D. Boneh, E. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. Theory Cryptogr. Conf.*, 2005, pp. 325–341.

[27] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, 2016.

[28] S. Carielli and L. Sustar (Forrester Res. Inc., Cambridge, MA, USA). *Best Practices: Kubernetes Security—Coordinate Across Identity, Network, and Container Security to Protect K8S*. (Jun. 2023). [Online]. Available: https://reprints2..com/#/assets/2/1941/RES179415/report

[29] E. Carmona-Cejudo, F. Iadanza, and M. S. Siddiqui, "Optimal offloading of Kubernetes pods in three-tier networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2022, pp. 280–285.

[30] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–37, Dec. 2022.

[31] M. Chase, "Multi-authority attribute based encryption," in *Proc. 4th Theory Cryptogr. Conf.*, 2007, pp. 515–534.

[32] N. Chen, J. Li, Y. Zhang, and Y. Guo, "Efficient CP-ABE scheme with shared decryption in cloud storage," *IEEE Trans. Comput.*, vol. 71, no. 1, pp. 175–184, Jan. 2022.

[33] S. Chen, J. Li, Y. Zhang, and J. Han, "Efficient revocable attribute-based encryption with verifiable data integrity," *IEEE Internet Things J.*, early access, Oct. 23, 2023, doi: 10.1109/JIOT.2023.3325996.

[34] A. Chiquito, U. Bodin, and O. Schelén, "Attribute-based approaches for secure data sharing in industrial contexts," *IEEE Access*, vol. 11, pp. 10180–10195, 2023.

[35] *Regulation (EU) 2016/679 of the European Parliament and of the Council*, European Parliament, Brussel, Belgium, 2016

[36] F. Faticanti et al., "Distributed cloud intelligence: Implementing an ETSI mano-compliant predictive cloud bursting solution using openstack and Kubernetes," in *Proc. 17th Int. Conf. Econ. Grids, Clouds, Syst., Serv.*, 2020, pp. 80–85.

[37] B. Ghosh. "Hybrid Kubernetes model: Bridging clouds and on-premises." Medium. Sep. 2017. [Online]. Available: https://medium.com/@bijit211987/hybrid-kubernetes-model-bridging-clouds-and-on-premises-4206cd430d50

[38] B. Ghosh. "The evolution of Kubernetes clusters in multicloud and hybrid cloud." Medium. Apr. 2023. [Online]. Available: https://addozhang.medium.com/the-evolution-of-kubernetes-clusters-in-multi-cloud-and-hybrid-cloud-b243aa7a4eab

[39] C. Gocul. "Cloud bursting with virtual kubelet and KIP (kloud instance provider)." LinkedIn. May 2020. [Online]. Available: https://www.linkedin.com/pulse/cloud-bursting-virtual-kubelet-kip-kloud-instance-provider-chandra/

[40] T. Guo, U. Sharma, T. Wood, S. Sahu, and P. J. Shenoy, "Seagull: Intelligent cloud bursting for enterprise applications.," in *Proc. USENIX Annu. Tech. Conf.*, 2012, pp. 361–366.

[41] Y. Guo, Z. Lu, H. Ge, and J. Li, "Revocable blockchain-aided attribute-based encryption with escrow-free in cloud storage," *IEEE Trans. Comput.*, vol. 72, no. 7, pp. 1901–1912, Jul. 2023.

[42] M. A. Ibrahim, G. A. Ebrahim, and H. K. Mohamed, "A modern cloud bursting framework," in *Proc. 12th Int. Conf. Comput. Eng. Syst. (ICCES)*, 2017, pp. 148–153.

[43] B. Kar, W. Yahya, Y. Lin, and A. Ali, "Offloading using traditional optimization and machine learning in federated cloud–edge–fog systems: A survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 1199–1226, 2nd Quart., 2023.

[44] C. Lan, L. Liu, C. Wang, and H. Li, "An efficient and revocable attribute-based data sharing scheme with rich expression and escrow freedom," *Inf. Sci.*, vol. 624, pp. 435–450, May 2023.

[45] J. Lee, S. Oh, and J. W. Jang, "A work in progress: Context based encryption scheme for Internet of Things," *Procedia Comput. Sci.*, vol. 56, pp. 271–275, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915016890

[46] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proc. 30th Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2011, pp. 568–588.

[47] J. Li, W. Yao, Y. Zhang, H. Qian, and J. Han, "Flexible and fine-grained attribute-based data storage in cloud computing," *IEEE Trans. Services Comput.*, vol. 10, no. 5, pp. 785–796, Sep.–Oct. 2017.

[48] J. Li et al., "Multiauthority attribute-based encryption for assuring data deletion," *IEEE Syst. J.*, vol. 17, no. 2, pp. 2029–2038, Jun. 2023.

[49] H. Lu, R. Yu, Y. Zhu, X. He, K. Liang, and W. Cheng-Chung Chu, "Policy-driven data sharing over attribute-based encryption supporting dual membership," *J. Syst. Softw.*, vol. 188, Jun. 2022, Art. no. 111271.

[50] (Microsoft Corp. Technol. Corp., Redmond, WA, USA). *Introduction to Azure Kubernetes Service (AKS)*. Accessed: Jun. 8, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/aks/intro-kubernetes

[51] A. Mosteiro-Sanchez, M. Barcelo, J. Astorga, and A. Urbieta, "Too many options: A survey of abe libraries for developers," 2022, *arXiv:2209.12742*.

[52] C. Padro, "Lecture notes in secret sharing," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2012/674, 2012.

[53] X. Qin, Z. Yang, Q. Li, H. Pan, Z. Yang, and Y. Huang, "Attribute-based encryption with outsourced computation for access control in IoTs," in *Proc. 3rd Asia Serv. Sci. Softw. Eng. Conf.*, 2022, pp. 66–73.

[54] M. Rengo. "Project onehundredten: Cloud bursting approaches based on Kubernetes ." GitHub. Accessed: Mar. 29, 2023. [Online]. Available: https://github.com/MRColorR/onehundredten.git

[55] M. Repetto, D. Striccoli, G. Piro, A. Carrega, G. Boggia, and R. Bolla, "An autonomous cybersecurity framework for next-generation digital service chains," *J. Netw. Syst. Manage.*, vol. 29, no. 4, p. 37, 2021.

[56] S. Risco, G. Moltó, D. M. Naranjo, and I. Blanquer, "Serverless workflows for containerised applications in the cloud continuum," *J. Grid Comput.*, vol. 19, no. 3, Jul. 2021.

[57] R. Ronan. "Simplify your hybrid/multi-cluster, multi-cloud Kubernetes deployments with KubeSlice." Developers Blog. Apr. 2022. [Online]. Available: https://blogs.oracle.com/developers/post/simplify-your-hybridmulti-cluster-multi-cloud-kubernetes-deployments-with-kubeslice

[58] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proc. 24th Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, Aarhus, Denmark, 2005, pp. 457–473.

[59] S. Sciancalepore, G. Piro, D. Caldarola, G. Boggia, and G. Bianchi, "On the design of a decentralized and multiauthority access control scheme in federated and cloud-assisted cyber-physical systems," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 5190–5204, Dec. 2018.

[60] V. Sharma, "Managing multi-cloud deployments on Kubernetes with Istio, rometheus and Grafana," in *Proc. 8th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, 2022, pp. 525–529.

[61] O. Tomarchio, D. Calcaterra, and G. D. Modica, "Cloud resource orchestration in the multi-cloud landscape: A systematic review of existing frameworks," *J. Cloud Comput.*, vol. 9, pp. 1–24, Sep. 2020.

[62] S. Touw (Immuta Inc., Boston, MA, USA). *Role-Based Access Control Vs. Attribute-Based Access Control*. (Apr. 2023). [Online]. Available: https://www.immuta.com/blog/attribute-based-access-control/

[63] M. Venema, G. Alpár, and J. Hoepman, "Systematizing core properties of pairing-based attribute-based encryption to uncover remaining challenges in enforcing access control in practice," *Des., Codes Cryptogr.*, vol. 91, no. 1, pp. 165–220, Sep. 2023.

[64] Z. Wan, J. Liu, and R. H. Deng, "Hasbe: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing," *IEEE Trans. Inf. Forensics Security*, vol. 7, pp. 743–754, 2012.

[65] B. Waters, "Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions," in *Proc. 29th Annu. Int. Cryptol. Conf.*, 2009, pp. 619–636.

[66] R. Zhang, J. Li, Y. Lu, J. Han, and Y. Zhang, "Key escrow-free attribute based encryption with user revocation," *Inf. Sci.*, vol. 600, pp. 59–72, Jul. 2022.

[67] Y. Zhuang (Alibaba Cloud Comput. Softw. Com., Hangzhou, China). *Enhancing Self-Created Kubernetes With Cloud Elasticity to Cope With Traffic Bursts*. (Oct. 2023). [Online]. Available: https://www.alibabacloud.com/blog/enhancing-self-created-kubernetes-with-cloud-elasticity-to-cope-with-traffic-bursts_600491

MAURO FEMMINELLA received the master's and Ph.D. degrees in electronic engineering from the University of Perugia in 1999 and 2003, respectively, where he has been an Associate Professor with the Department of Engineering since July 2022. He is also the Representative of the University of Perugia in consortium CNIT. He is a coauthor of more than 120 papers in international journals and refereed international conferences. His current research interests focus on molecular communications, big data systems, and network management solutions for 5G/6G networks.

MARTINA PALMUCCI was born in Jesi, Italy, in 1997. She received the bachelor's degree in mathematics from the University of Camerino, Italy, in 2020, and the master's degree in computer engineering from the University of Perugia, Italy, in 2022.

She broadened her academic exposure through Erasmus programs with the University of Salamanca, Spain, in 2017 and the Vrije Universiteit Brussel, Belgium, in 2022. She currently works as a Research Fellow with the GARR Consortium, Rome, Italy. Her contributions center on the exploration of cryptography, reflecting her unwavering commitment to enhancing cybersecurity practices. Her primary focus lies in applied cryptography and cybersecurity.

GIANLUCA REALI received the Ph.D. degree in telecommunications from the University of Perugia in 1997, where he has been an Associate Professor with the Department of Engineering since January 2005. From 1997 to 2004, he was a Researcher with the Department of Electronic and Information Engineering, University of Perugia. In 1999, he visited the Computer Science Department, UCLA. His research activities include resource allocation over packet networks, wireless networking, network management, multimedia services, big data management, and nanoscale communications. He is also a member of CNIT.

MATTIA RENGO was born in Terni, Italy, in 1995. He received the bachelor's degree in computer and electronic engineering with a focus on the Computer Science curriculum from the University of Perugia in 2020, and the master's degree in computer and robotics engineering, specializing in data science, from the University of Perugia in 2023.

Currently working as a DevOps Specialist, he is also active on GitHub. Some of his personal projects have garnered more than a hundred stars, showcasing their impact and usefulness in the tech community. His professional and academic commitments reflect his dedicated focus on computer engineering, particularly in cloud technologies and data science.

Open Access funding provided by 'Università degli Studi di Perugia' within the CRUI CARE Agreement