Sharing Data Through Confidential Clouds: An Architectural Perspective

Daniele Sgandurra*, Francesco Di Cerbo†, Slim Trabelsi†, Fabio Martinelli‡, Emil Lupu*
*Department of Computing, Imperial College London, UK
Email: {d.sgandurra, e.c.lupu}@imperial.ac.uk
†Product Security Research, SAP Labs, France
Email: {francesco.di.cerbo, slim.trabelsi}@sap.com
‡IIT-CNR, Via G. Moruzzi 1, I-56100 Pisa, Italy
Email: fabio.martinelli@iit.cnr.it

***Abstract*—Cloud and mobile are two major computing paradigms that are rapidly converging. However, these models still lack a way to manage the dissemination and control of personal and business-related data. To this end, we propose a framework to control the sharing, dissemination and usage of data based on mutually agreed Data Sharing Agreements (DSAs). These agreements are enforced uniformly, and end-to-end, both on Cloud and mobile platforms, and may reflect legal, contractual or user-defined preferences. We introduce an abstraction layer that makes available the enforcement functionality across different types of nodes whilst hiding the distribution of components and platform specifics. We also discuss a set of different types of nodes that may run such a layer.**

I. Introduction

Outsourcing data storage and management to the Cloud, usually for improved availability and sharing, entails an inherent loss of control by Cloud tenants. Therefore, additional means of data protection and usage control must be provided to allow organizations to safely share their data on the Cloud. Protecting data, and in particular personal data in the sense of European and National Legislations[1], is essential to citizens and organizations across all sectors, including government, healthcare and public administrations. In fact, data sharing between individuals and organizations through the Cloud can only be achieved by providing assurances on data protection. Furthermore, the data originator should be able to retain control over data usage to avoid uses for different, and unanticipated, purposes.

The goal of our project[2] is to provide a framework for information sharing among organizations that permits confidential and secure operations, with a strong emphasis on legal and regulatory aspects, and that is convenient for end-users. Currently, many organizations share data in an *ad hoc* fashion (e.g., by e-mail) or with contracts that are not automatically enforced an/or that rely on good behavior of the recipients. With our framework, organizations can effectively deploy contracts to share data with users, and other organizations, and automatically enforce the requirements and constraints over their subsequent use. In our architecture, an electronic *Contract*, or *Data Sharing Agreement* (DSA), is a formal document that regulates how organizations and/or individuals share data. In this paper we introduce an uniform enforcement component that seamlessly enforces DSAs through Cloud services and mobile devices. We detail how this component behaves as an abstraction layer that makes available the DSA enforcement functionality across different types of nodes, while at the same time hiding the distribution of components and platform specifics.

The paper is structured as followed. In Sect. II we describe the architecture and its main components, whereas In Sect. III we detail the layer that implements the enforcement and publishing functionality on different types of Cloud and mobile nodes. Section IV describes the types of nodes that can participate in the framework. In Section V we discuss related work. Finally, in Sect. VI we conclude the paper.

II. Architecture

In this section we give a high-level description of the proposed architecture, by describing the rationale, its main structure, and how it caters for different types of nodes, and how it interacts with external and legacy services. One of the key requirements for the framework is that it should cater for a variety of access and storage modalities for the information. In particular, Coco Cloud supports several deployment and service models, since its services may be offered in a public, private, or hybrid Cloud, and in IaaS or SaaS services. To this end, Coco Cloud makes available several typologies of nodes used to instantiate the framework according to the chosen models and the business needs. Across these different types of nodes, and their combinations, the framework provides access and distribution transparency as well as transparent enforcement of data protection. Hence, it must abstract out from the specifics of each platform and encapsulate distribution and coordination of enforcement in an abstract layer that provides uniform access to its functionalities.

We thus introduce an abstraction layer (called *Enforcement & Publishing layer*, see Fig. 1) as a uniform component that offers an entire set of enforcement functionalities for the DSAs on each node regardless of its characteristics. The implementation of the layer on each different type of node

[1] "Opinion 4/2007 on the concept of personal data", available at: http://ec.europa.eu/justice/policies/privacy/docs/wpdocs/2007/wp136_en.pdf

[2] http://www.coco-cloud.eu/

may vary: some of the services may be implemented locally, others may be implemented by having recourse to remote services, such as in a SaaS implementation. For example, information created and published on a mobile node may need to be analyzed remotely to be associated with the correct DSA. Interfacing of nodes with local and remote storage, e.g. a Cloud storage with S3 protocol, also happens at this level. Such a layer focuses on the enforcement of the policies resulting from the DSA, including both access rights to the data and obligations associated with data usage. These policies are enforceable in the sense that they can be interpreted by automated components.

Concerning Cloud nodes, according to chosen service and deployment model, we distinguish between: (i) *Cloud as applications and storage* nodes, for access and use of the data: in Fig. 1, this is the third node on the left, which is instantiations of a generic Coco Cloud node that includes both applications to access the protected data and also a storage space (also mobile and desktop/laptop nodes have similar requirements); *Cloud as a storage service* nodes: in Fig. 1, the rightmost node, which is an instantiation of a public provider storage node where the content of protected data is not processed nor accessed by provider applications, and data are only stored and replicated.

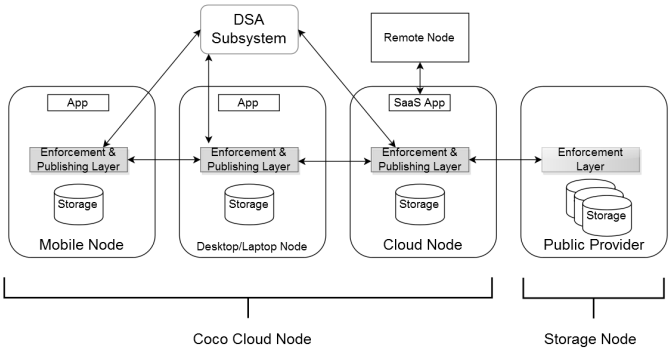

Figure 1. Overview of the Architecture

This distinction is important because it relates to the different types of access, enforcements and threats. In particular, a Coco Cloud node containing applications needs to access the content of the data exchanged, may change it, create new data, share or disseminate them further. In this case, the layer mediates between these applications and the protected resources by enforcing all policies on content access specified in the DSA. Examples of Coco Cloud nodes, as shown in Figure 1, include: (i) mobile node: this is any mobile device that includes the layer; (ii) desktop/laptop node: this is a standard desktop or laptop computer that includes the layer; (iii) Cloud node: this is a SaaS node, which is accessed remotely via a specific interface by a client node, and that includes the layer to access its functionalities.

In the second set of nodes, storage nodes, a framework-aware provider does not need to access the information content, and it is exploited for its storage capabilities only. In these cases, a public provider is treated as an *honest-but-curios provider*, i.e. one provider willing to provide a set of enforcement functions, without modification, but at time may be tempted to access confidential data. Such a (storage) provider still needs to enforce those policies that relate to the container in which the data is encapsulated. For example, it may need to delete the container under a data retention or privacy policy, record any copies or which clients have downloaded it.

We consider the data content to be encapsulated in a *Data Protected Object* (DPO) encrypted with a content key (or a reference to it, e.g. to a trusted authority) and also containing other metadata, such as the policies relating to data usage. In case of CASS nodes, because access to the information content is not required, access to the DPO content itself is also not needed. However, the meta-data and policies on the container management may also be confidential, so DPOs are included into a higher level container called a *Data Object Container* (DOC), which may be encrypted in turn.

When a node wants to protect some data through DSAs, a new object is created through the enforcement layer. The relevant DSAs are attached to the object as enforceable policies. The created object has a layered structured where all, or a subset of, its DSAs can be understood and enforced by the nodes participating in the framework. This layered structure depends on, among the others, node location, node type etc. Usage control requires that the applications be certified and trusted (i.e., their integrity checked) to access raw data. However, the architecture is flexible enough to define several different kinds of layered policies format.

III. Enforcement and Publishing Layer

The enforcement and publishing layer is responsible for (i) the enforcement of DSA (through the *enforcement subsystem*) or (ii) the publishing of new object(s) under DSAs (through the *publishing service*). The protected objects can be stored locally or remotely through the *Storage Adapter*, or sent to another node of the architecture.

To interact with the enforcement and publishing layer, there are two possibilities. In the first one, a framework-aware application is used, e.g. one that knows how to invoke the functionalities of the enforcement & publishing layer, which are exported through a set of layer API. This interaction is possible because when such an application is developed it is linked against a library that allows the application to invoke the layer functions natively. In particular, the API is used to invoke the functions of the framework layer. As an example, this set of API includes the function to publish a protected object, to request access to a protected object, to store a protected object, etc. In the second case, a legacy application is interfaced with the framework layer through a protocol/application adapter, whose goal is to convert the invocations/requests of the application (e.g., to read or to store a file) to requests to the layer using its exported API. As an example, the adapter has to append some metadata to the data to be published through the enforcement & publishing layer or

to invoke the functions to enforce the DSAs when accessing a protected file. This adapter is logically not a part of the layer, since it is a distinct component whose only function is to translate, if needed, application's request in native format into layer-compliant requests.

A. Enforcement Subsystem

A fundamental component of the framework layer is the *enforcement subsystem*. The enforcement subsystem is responsible of enforcing the attached DSA policies on either DOP or DOC. Examples are enforcement of obligations, continuous authorizations and managing mutable attributes (e.g., attribute which counts a number of data copies in the system). In contrast with a traditional access control architecture, which considers all access requests as short lived and atomic, usage control also constrains access in terms of dynamically changing attributes and needs to interface with local context services through a publish/subscribe mechanism, which allows the enforcement to receive notification whenever some events have occurred (e.g., a period of time has elapsed). In the framework, different nodes may be trusted to a different degree and, on some platforms, the integrity of the enforcement may be a crucial factor in governing the access. In particular, data should not be accessed (or no longer be accessed) if there are reasons to believe that the integrity of the node, or of the enforcement system, has been compromised.

B. Publishing Service

The other important functionality of the framework layer is the *Publishing Service*, which is the component of the framework used to publish new objects (files) protected under the relevant DSAs. To publish protected objects, the layer needs to understand which DSAs need to be associated with them. This can be either chosen by the user (i.e., a discretionary decision) or it might be based on attributes of the object itself (i.e., a mandatory decision), or the context in which the object has been created (e.g., which application, in which folder, at which point in time). When such a protected object is created, i.e. a protected object containing data and DSA (also called a *bundle*), it is ready to be consumed on any other enabled node. A storage adapter is used to store the bundle either locally, on an external storage or sent to another framework node.

C. Storage Adapters

An adapter is used for converting a request from a format into another one. In particular, adapters may reside outside the enforcement & publishing layer, such as the Protocol/Application Adapter, or inside the layer, such as the *Storage Adapter*. This adapter is the component used to store, and retrieve, protected object regardless of their location, which can be local, a Cloud storage or a remote node. To this end, the storage adapter is responsible of invoking the proper functions, e.g. system calls to save the object locally, S3/Swift API calls to save the object on an external Cloud storage or layer API calls to send the file to another framework node.

IV. Types of Nodes

As we have described previously, a generic node is any node that participates in the framework that is capable of running the enforcement & publishing layer to access and publish files. In a typical scenario, a user of an organization exploits the framework to share data with several other organizations with which it has drawn a set of multilateral DSA. This user can access, or publish, documents protected under such DSAs by either using a legacy application or an application aware of the framework. There are at lest three types of node that can exploit the framework, which will be detailed in the following.

Mobile Node: An instantiation of a generic node can be any mobile device capable of running the enforcement & publishing layer. For example, it can be a corporate mobile device, i.e. belonging to the organization and as such more controlled, or personal devices used as a work tool within organizations adopting the "bring your own device" (BYOD) paradigm. In these scenarios, the layer is responsible of providing strong assurance that the BYOD devices cannot be compromised and that the access to the data cannot be given without checking DSA policies (e.g., by "turning off" some components, such as the enforcement subsystem). To this end, a pluggable component of the layer, called *Integrity Manager*, is responsible, when requested by the threat model, of checking the integrity of the system. The goal of this component is to verify that the system, and the layer itself, are running untampered with. In this cases, the Integrity Manager might exploit techniques such as remote attestation [1], with the help of Trusted Platform Module (TPM) [2], TrustZone technologies [3], in combination with Mobile Device Management (MDM), to create a chain of measurements.

Desktop/Laptop Node: In this scenario, a desktop or a laptop is used to access the framework functions, e.g. to create new objects or to be able to access existing protected objects. Usually, either legacy applications are used to access the protected objects or *ad hoc* applications might be created to interface with the layer (e.g., a plugin for a Web Browser). In these cases, the desktop and laptop nodes are under strict control of the organization, e.g. with respect to what software can be installed, who can access the devices, etc., and hence the security requirements are less strict then the mobile case, although also in this scenario a component to check the integrity of the node (Integrity Manager) might be included, and exploited, if the threat model requires it.

SaaS Cloud Node: In this scenario, the enforcement & publishing layer is running on a Cloud node (either belonging to a private or a public Cloud infrastructure), and also the applications that need to interact with other nodes of the architecture, or with protected objects, run on the Cloud node itself. Basically, this scenario describes a SaaS node and, as in most of these cases, the Cloud node is typically accessed by users using a remote client through a set of exported APIs (e.g., REST). The threat model in these cases, and the corresponding use of the Integrity Manager, is related to the trust associated with the Cloud Provider hosting the SaaS application.

Storage Node: When a protected object needs to be stored (e.g., when it has been published), it can be stored on a remote Cloud storage, for example to be shared between two or more organizations. The storage node, i.e. the Cloud service provider, is *framework-aware*, which means that it runs a (stripped) version of the enforcement & publishing layer able to cope with DOC policies only. The functionalities of the layer in an aware storage node are limited, as an example, on enforcing DSA on DOC objects, such as policies on geography placement, validity time, and similar ones. In contrast, the layer does not need to include functionalities to publish or access the object directly, since this is only a storage node. This scenario caters for *honest-but-curious provider*, i.e. a provider who (1) stores the outsourced protected objects without tampering with their integrity; (2) honestly executes every framework operation and returns protected objects on request; (3) may try to access user's data, inside a protected object, improperly. In a nutshell, this scenario depicts a provider that respects the integrity of the objects (and of the enforcement) but may not respect the confidentiality of the data.

V. Related Work

Within the FP7 Consequence Project an enhanced enforceable policy language has been defined. This environment has been further extended to enforcement on TPM platforms and use of attestation in the policy conditions [4] [5]. However, much of the focus has been on the enforcement for data dissemination in crisis management scenarios where issues of intermittent connectivity must be addressed. To this end a novel policy-based trust management framework has been defined that allows recursively devolving authority for policy and attribute evaluation [6]. *DataSafe* [7] is an architecture that uses policies to protect data from attacks from untrusted third-party application, such as untrusted third party applications. To this end, DataSafe provides dynamic instantiations of secure data compartments and continuously tracks and propagates hardware tags to identify sensitive data by enforcing unbypassable output control. *Guardat* [8] is an architecture that enforces data access policies at the storage layer, where the controller can be implemented in a trustlet isolated using a protected container with Intel SGX.

VI. Conclusion

The challenges of sharing data, whilst preserving some degree of control over its usage, can be met by extending and integrating techniques from Enterprise Rights management, policy-based access and usage control, key management and trust management. Explicit and automatically enforceable Data Sharing Agreements are necessary to establish the trust relationships between the parties, their expectations, rights and duties which translate into the usage control policies to be enforced. However, the integration of the techniques involved in their enforcement and catering for the wide spectrum of devices and services involved into a common, extensible and modular architecture, is a significant challenge. A common middleware layer is necessary, not only to hide the distribution aspects, but also to hide how services and functionality is brought to the point of use. This enables the layer to transparently offer all the functionalities on all devices even if some of these may be offered remotely, and to abstract from how the services are provided. Data are protected within containers and associated with policies. Different policies may apply to the management of an opaque (i.e., encrypted) container, and then to the usage of the data, each needing to be enforced by a different provider. This leads to the encapsulation of policy layers, where each layer contains those policies that apply to the management of its immediate contents.

In this paper we have described the design principles of an architecture to control the sharing, dissemination and usage of data on Cloud. We have described the enforcement & publishing layer, which is the component responsible for creating objects and enforcing the policies on the protected objects. Furthermore, we have shown how such a layer can be instantiated onto different types of node. Currently, the architecture is being validated on three Pilot implementations, catering for three industry verticals, each offering a wide range of use-cases.

Acknowledgment

The research leading to these results has received funding from the FP7 EU-funded project Coco Cloud (grant No. 610853).

References

[1] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen, "Principles of remote attestation," *Int. J. Inf. Secur.*, vol. 10, no. 2, pp. 63–81, Jun. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10207-011-0124-7

[2] S. Pearson, "Trusted computing platforms, the next security solution," *HP Labs*, 2002.

[3] J. Winter, "Trusted computing building blocks for embedded linux-based arm trustzone platforms," in *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing*, ser. STC '08. New York, NY, USA: ACM, 2008, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/1456455.1456460

[4] V. Gowadia, E. Scalavino, E. C. Lupu, D. Starostin, and A. Orlov, "Secure cross-domain data sharing architecture for crisis management," in *Proceedings of the Tenth Annual ACM Workshop on Digital Rights Management*, ser. DRM '10. New York, NY, USA: ACM, 2010, pp. 43–46. [Online]. Available: http://doi.acm.org/10.1145/1866870.1866879

[5] A. Gopalan, V. Gowadia, E. Scalavino, and E. Lupu, "Policy driven remote attestation," in *Security and Privacy in Mobile Information and Communication Systems*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, R. Prasad, K. Farkas, A. Schmidt, A. Lioy, G. Russello, and F. Luccio, Eds. Springer Berlin Heidelberg, 2012, vol. 94, pp. 148–159. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30244-2_13

[6] E. Scalavino, V. Gowadia, R. Ball, E. C. Lupu, and G. Russello, "Mobile paes: Demonstrating authority devolution for policy evaluation in crisis management scenarios," *Policies for Distributed Systems and Networks, IEEE International Workshop on*, vol. 0, pp. 53–56, 2010.

[7] Y.-Y. Chen, P. A. Jamkhedkar, and R. B. Lee, "A software-hardware architecture for self-protecting data," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 14–27. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382201

[8] A. Vahldiek, E. Elnikety, A. Mehta, D. Garg, P. Druschel, A. Post, R. Rodrigues, and J. Gehrke, "Guardat: A foundation for policy-protected data," MPI-SWS, Tech. Rep. 014-002, MPI-SWS, 2014.