Adaptive Data Management for Self-Protecting Objects in Cloud Computing Systems

Anna Cinzia Squicciarini[1], Giuseppe Petracca[1], Elisa Bertino[2]
1 College of Information Sciences and Technology, Pennsylvania State University
2 Computer Science Department, Purdue University

***Abstract*—While Cloud data services are a growing successful business and computing paradigm, data privacy and security are major concerns. One critical problem is to ensure that data owners' policies are honored, regardless of where the data is physically stored and how often it is accessed, and modified. This scenario calls for an important requirement to be satisfied. Data should be managed in accordance to owners' preferences, Cloud providers service agreements, and the local regulations that may apply. In this work we propose innovative policy enforcement techniques for adaptive sharing of users' outsourced data. We introduce the notion of autonomous security-aware objects, that by means of object-oriented programming techniques, encapsulate sensitive resources and assure their protection. Our evaluation demonstrates that our approach is effective.**

I. Introduction

Cloud computing services enable individuals and organizations to outsource data management with ease and at low cost. However, while Cloud data services are a growing successful business and computing paradigm, data privacy and security are major concerns. Data stored in the Cloud often encodes sensitive information and should be protected as mandated by organizational policies and legal regulations. A commonly adopted approach is to encrypt the data when uploading them to the Cloud. Encryption alone however is not sufficient as often organizations have to access, modify and selectively share data with other parties [13], [14]. Policies must be associated with the data to specify which party can access the data for performing which actions. Such policies can be quite complex as they typically encode different access constraints. Furthermore, a Cloud secure data management system should be able to demonstrate ongoing conformity with their customer's business compliance and regulatory requirements (depending on industry and jurisdiction). Maintaining separate compliance efforts for different regulations and standards in an automated fashion is knowingly a major challenge in the Cloud domain. Access restrictions for sensitive data have to adapt not only to changes by the content originators (or owners), but also to privacy and auditing regulations that are in place in the location where data is stored and managed. Furthermore, depending on the location, different architectures and standards may affect data representation and management, resulting in compatibility issues.

Toward addressing these issues, in this work we propose innovative policy enforcement techniques for adaptive sharing of users' outsourced data. We introduce the notion of *autonomous security-aware objects* (SAOs), that by means of object-oriented programming techniques, encapsulate sensitive resources and assure their protection. Such protection is afforded through the provision of adaptive security policies of various granularity. The resources protected by SAOs may be files of different kind, including images, text, and media content. The security policies can include constructs to specify access control, authentication and usage controls. Furthermore, they are enforced according to contextual criteria to ensure compliance with location-specific regulations and service level agreements. Through careful design and deployment, SAOs provide strong security guarantees, and adapt to local compliance requirements. As part of our solution, we show how SAOs can either use locally pre-loaded policies or securely accept new policies from trusted authorities.

We have implemented a running prototype of the proposed approach using Java™-based technologies [1], [11]. We have extensively tested our prototype, and used the Amazon Web Services (AWS) APIs to demonstrate how our solution can be integrated with actual CSPs.

The rest of the paper is organized as follows. Next section provides an overview of the notion of SAO. Section III discusses the security policies and their enforcement. Section IV presents the deployment of our architecture and discusses the context retrieval process. Section V reports experimental results. Section VI discusses related work and Section VII concludes the paper.

II. Overview of the SAO-based solution

A. Overview of Secure-Aware Objects

Any object-oriented approach designed to achieve adaptive data protection has to address the following requirements: protection of encapsulated data through enforcement of security policies associated with the data, interoperability, portability, and object security. Our SAO-based approach addresses the first requirement by provisioning owner-specified security policies that are tightly bound with the content, and executable within the SAO. To ensure interoperability, our SAOs are self-contained, and do not require any dedicated software to execute, other than the Java running environment. Precisely, each SAO includes five key components: (1) authentication and authorization tools; (2) self-enforcement policy engine; (3) security policies in executable form; (4) secure connections manager; and (5) protected file(s);

978-3-901882-48-7

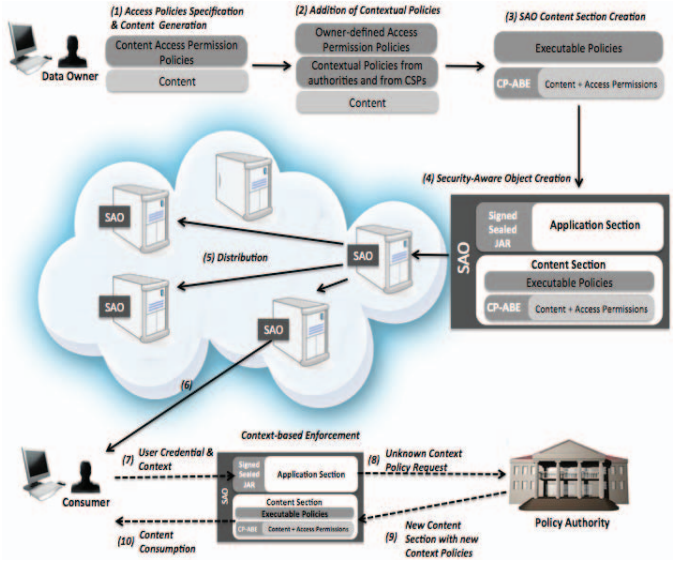

Fig. 1. Overall Approach

Self-containment ensures portability, in that SAOs may be moved and replicated without the need of installing dedicated software. We employ nested Java Archives (JARs) [11] to achieve both the portability of the protected content and the portability of the modules required to access and use the content as intended by its originator.

In order to guarantee adaptivity to the context from where the content is accessed, the policies embedded in the SAO are sensitive to the location and other contextual dimensions that may affect the enforcement process.

Finally, object security is guaranteed by advanced cryptographic primitives, such as Ciphertext-Policy Attribute-Based Encryption [4] and Oblivious Hashing for program tracing [7], combined with extended and advanced object-oriented coding techniques. We leverage built-in Java constructs, like Java policies and Java Authentication and Authorization Service [1], to implement the object's content protection mechanisms.

B. Interaction Flow

Our SAO-centered architecture deals with four different classes of stakeholders, that is, data owners, government authorities, cloud service providers (CSPs), and content consumers. These entities are in charge of authoring policies for the protected content, and/or accessing the content upon need. In particular, government authorities issue regulations concerning content sharing and protection, and require that these regulations be enforced by organizations managing data; whereas CSPs may enforce rules reflecting their terms of service or to facilitate data processing. data owners may have their own personal policies concerning the access and use of their own content. Finally, content consumers access content, their accesses must comply with all the access restrictions imposed on the content by the other stakeholders.

In what follows, we assume that government authorities and service providers are represented by a trusted third-party, referred to as *Policy Authority* (PA), which is in charge of storing high-level regulations and service level agreements, and translate them in policy rules expressed in a machine interpretable form. The generated policy rules can thus be automatically enforced by a computer system.

Figure 1 shows the interplay among the four stakeholders during generation and consumption of SAO. In detail, a SAO is generated by a content originator subscribed to a CSP. The generation of a SAO includes three major phases, reported on top in the figure. The first phase is the specification by the content originator of policy rules, for controlling the sensitive content (e.g. text, video or image file) to be stored at the CSP (step 1). The second phase is the specification by the policy authorities of policy rules whose enforcement assures compliance with security and data disclosure regulations (e.g. auditing requirements), as well as with SLAs specified by cloud service providers (step 2). The third phase is the compilation of all these policy rules into executable policies. The executable policies are then encrypted and sealed with the content using CP-ABE primitives (step 3). After these three phases, the generation of the SAO is completed.

Once created, the content stored in each SAO may be accessed according to its security policy. SAO consumers may include authorized users, CSPs for processing and analysis, and auditors/authorities to meet compliance requirements. Each time access to the SAO content is attempted, its policy rules are evaluated for applicability, according to the requested subject credentials, and contextual features (step 7).

The policy protecting the content stored within the SAO is organized in rules of varying priority, and each rule may be context-specific. As the SAO changes location, high-priority rules may be added to the SAO, using a third-party invocation protocol if the rules are not stored within the SAO already. Specifically, each policy rules set generated by a local authority is stored in a repository managed by the local PA, as shown in steps 8 and 9 of Figure 1. The PA guarantees that each policy rule contained in the set is within the scope of the authority jurisdiction, and that it is applicable only for SAO stored within the authority's domain.

III. SAO Policies

In this section, we first introduce an abstract representation of the security policies supported by our system. Next, we discuss how these are translated within the SAO.

A. Policy Specification

We define a security policy as a collection of declarative rules, i.e. $P = \{r_1, \ldots, r_n\}, n \geq 1$. Each rule controls access and usage of a given content by a certain entity (e.g. user, service provider, government authority). It may either target a single content item f (e.g. file) stored within the SAO, or if the SAO stores multiple content elements (i.e. files), it may refer to all of them.

A rule r is a tuple composed by the following elements: $<$ `PolicyL:Effect`, `Act`, `Subj`, `Obj`, `Location`, `Time` $>$. Besides the `PolicyL : Effect`, and the `Obj`, the remaining components are defined in terms of boolean conditions against a set of pre-defined variables. Boolean conditions take the form

(a op a′), where a is the attribute name, a′ the attribute value, and op is a basic atomic operator, such as $>, <, =, \geq$, and $\leq$. We assume that boolean conditions are atomic. We briefly discuss each component in what follows:

- PolicyL denotes the policy level and can take one of the following values: Auth, CSP, or Owner; whereas Effect is a boolean variable that represents an access $Deny$ (0) or a $Grant$ (1). The PolicyL is assigned according to the policy author, and it dictates rules' hierarchical order. Auth refers to rules specified by government authorities, and take priority over rules defined by other parties. CSP refers to rules that can be defined by CSPs for managing the user's content. This is a secondary priority level. Owner denotes the lowest priority level and it is used by data owners to control access of their own content.
- Act denotes the admitted action, and is therefore content-specific. Act is instantiated if the rule effect is set to 1.
- Subj is the targeted entity. A subject can be identified by means of a combination of one or more joint (disjoint) boolean conditions against the attributes of the credential used for authentication purposes at the SAO. For example, if X.509 certificates are used, any condition against any of the attributes included in the credential to qualify the subject (e.g. Country, Name, and Age) can be used. In addition, the policy author can constrain the class of entities the rule applies to, by specifying a boolean condition against a special Type attribute, if not supported directly by the credential. This type refers to the entity performing the access, which could be one of the three stakeholders: Auth, CSP, or User. In particular, Auth refers to (regulatory/legal) authorities, CSP refers to cloud service providers, and User refers to end consumers.
- Obj refers to the target of the rule. It can refer to an individual file within the SAO, or the whole object, in which case Obj assumes the default value SAO.
- Location refers to conditions related to the location of the data consumption. Location-based conditions can refer to both geographical and cyber locations. They are both expressed at various levels of granularity, from a general country or state, to the specific machine where the object can be executed. In the current version of our policy language, locations can be specified using boolean conditions against predefined attributes. Examples include Country, State, City, IPaddress. To ensure correct evaluation of location information, we request that if City $\neq$ NULL $\vee$ State $\neq$ NULL, then Country $\neq$ NULL.
- Time is used to represent any possible time constraints. Temporal conditions are represented by means of finite intervals [3] $[t_b, t_e]$, where t_b and t_e denote the lower and upper bound of the interval, respectively. Open ended intervals are also supported (e.g. $t_b > 12/12/2014$).

B. Policy Rules Selection and Translation

When a new SAO is created or a SAO is moved to a new location, the policy composition and translation process is activated. This process is executed so as to ensure that only relevant and applicable rules are included in the SAO. Simply storing all the rules in the SAO may be impractical and not scalable. The policy translation process is preceded by a selection of the applicable rules, followed by a static ordering of applicability.

1) Policy rules selection: This first step identifies a set of applicable rules. Applicable rules are identified using current contextual information, without consideration of the subject requirements, as if subject information were not available at this point of time. Precisely, a policy rule r =<PolicyL : Effect, Act, Subj, Obj, Location, Time> is applicable to the current context c =< Country, State, City, IP, Timestamp>, if and only if: (a) the rule location conditions match the contextual information, that is c.Country = r.Country or c.State = r.State or c.City = r.City; (b) r refers to a file f witch matches or is included in the Obj component of the rule, i.e. Obj = f or Obj = SAO; (c) temporal restrictions are not obsolete, that is Timestamp $<$ r.Time.t_e.

Next, the applicable rules with highest priority are selected. To ensure correctness of execution, the rules are organized into classes according to the PolicyL values (i.e. Auth, CSP, Owner). The obtained sorted list is then used to ensure a deterministic order of evaluation of the policies, to be reflected at the time of policy translation.

2) Policy rules translation: The sorted list of rules are next translated into Java policies and access structures. Access structures are defined according to the specification of the CP-ABE encryption schema [4]. CP-ABE is in fact adopted as it supports the notion of attribute-based policies as a criteria for encryption, and is complementary to the Java policies. The access structures are embedded as part of the encrypted content, as by the CP-ABE construction. Which party is entitled to decrypt the content and under which context is addressed by means of the CP-ABE boolean access structure entries. Conversely, the Java policies are stored in the data container, i.e. the SAO, and specify which party can access a specific resource (code, files), and how. For example, the Java policies help render the content according to the specific access privilege.

A local Trusted Authority (TA) connected with the PA acts as key management authority, and is the only entity which knows and manages the cryptographic key needed for the encryption of data and the Master key needed for the creation of CP-ABE decryption keys associated with the users. We specifically employ the *Encrypt(PK, T, f)* primitive of the CP-ABE protocol for encryption, where T is the access structure representing attribute-based conditions (in terms of conjunctions and disjunctions), f is the file being encrypted, while PK is a set of public parameters.

T is implicitly contained in the ciphertext. T takes the form of a directed and acyclic graph. Upon evaluation T is traversed in depth, to ensure correct hierarchical ordering: the root node is $AUTH$, and the node $OWNER$ is last child node in the subtree with root *CSP*.

IV. SAO DEPLOYMENT AND CONTEXT RETRIEVAL

Architecturally, a SAO includes three components: the Application Section, the Content Section, and the Policy Section,

wrapped into an external JAR.

Application Section: a set of unmodifiable software modules, signed and sealed by the PA to ensure integrity of the application. The Application Section is itself deployed in a nested JAR, and organized into four components: Authentication and Access Control Component (AAC), Action Control Component (ACC), Content Protection Component (CPC), and the Connection Manager Component (CMC).

The AAC implements the authentication and authorization mechanisms, by exploiting the services offered by Java Authentication and Authorization Services (JAAS) [1] and Security Manager. By means of JAAS primitives, the SAO accepts X.509 certificates.

The ACC implements mechanisms for managing access to protected content, according to the outcome of the authentication and authorization process. This component accesses the Java policy file, which dictates what portions of the code to execute upon verification of identities and of authorization.

The CPC manages the protected content stored in the Content Section, as well as the protocols to re-deploy the new Content Section upon context change or time expiration. Content encryption and access control enforcement are integrated by means of the CP-ABE scheme [4].

The CMC handles all processes for creating secure connections toward the PA to receive the Content Section and toward the WorldWide Space and Time Services to retrieve the context information. The CMC component implements a Challenge-Handshake Authentication Protocol and exploits SSL connections to ensure private connections.

Content Section: stores the protected content items, the Java policy file that records the policies implementing the security rules, and an record with the last context in which the content has been accessed. Both Java policy file and the last context record are signed with the TA private key ($TA_{PrivKey}$) for integrity purposes. The Content Section represents the dynamic part of the SAO since it replaced by the TA, each time that the SAO is accessed in a new context or after a certain time period, to ensure compliance with more recent policy rules.

Policy Section: a collection of `owner` level policy rules, i.e., the set of policy rules specified by the data owner for the SAO. This section is encrypted with the user SK, thus only the TA is able to decrypt it.

V. Experimental Evaluation

Our evaluation testbed includes a client machine, a Mac Book Pro (with a 2.2 GHz Intel Core i7 process, and 4 GB 1333MHz DDR3 memory); and a server machine, an iMac (with a 3.06 GHz Core 2 Duo E8435 processor, and 8 GB 1333MHz DDR3 memory). The client machine acts as a content consumer, whereas the server machine hosts the TA. The client machine connects to the Amazon Web Services. In particular, we have used Amazon S3, Amazon EC2 and Amazon Cloud Drive for the storage and computing services.

We conducted several tests to evaluate the performance of our architecture. First, we evaluated the total time of reconstructing a SAO upon changing the context. We tested the reconstruction time by varying both the size of the SAO, and the size of the policy. We measured the size of the SAO in KB, ranging from 25 to 1000, and we measured the complexity of the policy in terms of size of the access structure, which complexity is defined in terms of nodes in the access structure. The time is not significantly affected by the file size, while there is an increase upon augmenting the size of the access structure. For a very complex policy, with 80 nodes, the reconstruction time is 2.6 seconds.

As a second experiment, we compared the delay in accessing a file with and without the SAO architecture. A file encrypted with the CP-ABE scheme (access structure with 25 nodes) is decrypted in 0.68 seconds. If the file is hosted by the SAO, the authentication, the context retrieval and the rendering procedures, added to the file, bring the access time to 1.31 seconds. A change of context adds further 1.0784 seconds to the earlier time, because in this case a communication and data exchange with the TA is required.

VI. Related Work

Access control [13], [15] and data management outsourcing techniques targeting the Cloud have been recently proposed [2], [6], [9]. Further, Cloud-specific cryptographic-based approaches for ensuring remote data integrity have been developed [14]. Subsequently, Wang and et al. [14] proposed a data outsourcing protocol specific to the Cloud. Wang's work is focused on auditing of stored data from trusted parties.

The notion of SAO is corroborated by previous projects [8], [5] and our own results [12], [10]. Our approaches are closely related to self-defending objects (SDO) [5] and self-protecting objects. SDOs are Java-based solutions for persistent protection to certain objects. SPOs are software components hosted in federated databases. The objective of SDOs is to ensure that all the policies related to any given object are enforced irrespective of which distributed database the object has been migrated to. SDOs depend on a Trusted Common Core for authentication and authorization, and therefore are not applicable in distributed systems. As a result, adaptiveness issues are not considered. Related to the idea of self-protecting data is also the work on sticky polices [8] that focuses on portability of disclosure policies by means of declarative policies tightly coupled to sensitive data by means of cryptographic algorithms. However, these policies are designed for federated organizations, and therefore lack adaptiveness to the domain of application. In previous work, we proposed an approach for accountability [10] in the Cloud.

VII. Conclusion

We presented an approach for secure and distributed data management in the Clouds. The idea behind our solution is to protect the data by means of wrappers applied to the targeted content file(s) which are "security-aware" in that they protect the content as it travels across domains by locally enforcing security policies. In the future, we plan to support more articulated access rights and contexts.

REFERENCES

[1] http://java.sun.com/products/archive/jaas/.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *ACM conference on Computer and Communications Security*, pages 598–609, 2007.

[3] E. Bertino, P. Bonatti, and E. Ferrari. Trbac: A temporal role-based access control model. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):191–233, 2001.

[4] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334, Washington, DC, USA, 2007.

[5] J. W. Holford, W. J. Caelli, and A. W. Rhodes. Using self-defending objects to develop security aware applications in java. In *27th Australasian conference on Computer science - Volume 26*, ACSC '04, pages 341–349, Darlinghurst, Australia, Australia, 2004.

[6] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard. A cooperative internet backup scheme. In *USENIX Annual Technical Conference*, pages 29–41, 2003.

[7] D. Liu and S. Xu. MuTT: A Multi-Threaded Tracer for Java Programs. In *8th IEEE/ACIS International Conference on Computer and Information Science*, pages 949–954, 2009.

[8] H. C. Pöhls. Verifiable and revocable expression of consent to processing of aggregated personal data. In *International Conference on Information and Communications Security (ICICS)*, pages 279–293, 2008.

[9] T. J. E. Schwarz and E. L. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *IEEE International Conference on Distributed Systems*, page 12, 2006.

[10] A. C. Squicciarini, S. Sundareswaran, and D. Lin. Preventing Information Leakage from Indexing in the Cloud. In *3rd IEEE International Conference on Cloud Computing*, 2010.

[11] Sun. Lesson: Packaging programs in jar files. http://java.sun.com/docs/books/tutorial/deployment/jar/.

[12] S. Sundareswaran, A. C. Squicciarini, D. Lin, and S. Huang. Promoting distributed accountability in the cloud. In *4th IEEE International Conference on Cloud Computing*, 2010.

[13] Q. Wang and H. Jin. Data leakage mitigation for discretionary access control in collaboration clouds. In *16th ACM symposium on Access control models and technologies*, SACMAT '11, pages 103–112, New York, NY, USA, 2011. ACM.

[14] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *ESORICS*, pages 355–370, 2009.

[15] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proceedings IEEE INFOCOM*, pages 1–9, 2010.