J Grid Computing (2017) 15:219–234
DOI 10.1007/s10723-017-9394-2

PaaSword: A Holistic Data Privacy and Security by Design Framework for Cloud Services

Yiannis Verginadis · Antonis Michalas · Panagiotis Gouvas · Gunther Schiefer · Gerald Hübsch · Iraklis Paraskakis

Received: 27 October 2015 / Accepted: 22 February 2017 / Published online: 22 March 2017

Abstract Enterprises increasingly recognize the compelling economic and operational benefits from virtualizing and pooling IT resources in the cloud. Nevertheless, the significant and valuable transformation of organizations that adopt cloud computing is accompanied by a number of security threats that should be considered. In this paper, we outline significant security challenges presented when migrating to a cloud environment and propose PaaSword – a novel holistic framework that aspires to alleviate these challenges. Specifically, the proposed framework involves a context-aware security model, the necessary policies enforcement mechanism along with a physical distribution, encryption and query middleware.

Keywords Data privacy · Security by design · Context-aware access control · Symmetric searchable encryption · Cloud computing

1 Introduction

Until recently, large-scale computing was available exclusively to large organizations with an abundance of in-house expertise. Cloud computing has changed that to the point where any user with even basic technical skills can obtain access to vast computing resources at low cost. In the technology adoption lifecycle, cloud computing has now moved from an early adopters stage to an early majority, where we typically see exponential number of deployments [37]. Throughout the past few years, many users have started relying on cloud services without realizing it. Major web mail providers utilize cloud technology;

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644814, the PaaSword project (www.paasword.eu) within the ICT Programme ICT-07-2014: Advanced Cloud Infrastructures and Services.

Y. Verginadis (✉)
Institute of Communications and Computer Systems, National Technical University of Athens, Athens, Greece
e-mail: jverg@mail.ntua.gr

A. Michalas
Cyber Security Group, University of Westminster, London, UK
e-mail: a.michalas@westminster.ac.uk

P. Gouvas
Ubitech Ltd., Athens, Greece
e-mail: pgouvas@ubitech.eu

G. Schiefer
Karlsruhe Institute of Technology, Karlsruhe, Germany
e-mail: gunther.schiefer@kit.edu

G. Hübsch
CAS Software AG, Karlsruhe, Germany
e-mail: gerald.huebsch@cas.de

I. Paraskakis
South East European Research Centre (SEERC),
The University of Sheffield, International Faculty,
CITY College, Thessaloniki, Greece
e-mail: iparaskakis@seerc.org

tablets and smartphones often default to automatically uploading user photos to cloud storage and social networks; finally, several prominent CRM vendors offer their services using the cloud. In other words, the adoption of cloud computing has moved from focused interest to widely spread intensive experimentation and is now rapidly approaching a phase of near ubiquitous use.

Enterprises increasingly recognize the compelling economic and operational benefits of cloud computing [31]. Virtualizing and pooling IT resources in the cloud enables organisations to realize significant cost savings and accelerates deployment of new applications, simultaneously transforming business and government at an unprecedented pace (CSA, 2013). However, those valuable business benefits cannot be unlocked without addressing new data security challenges posed by cloud computing.

Despite the benefits of cloud computing, many companies have remained cautious due to security concerns. Applications and storage volumes often reside next to potentially hostile virtual environments, leaving sensitive information at risk to theft, unauthorized exposure or malicious manipulation. Governmental regulation regarding data privacy and location presents an additional concern of significant legal and financial consequences if data confidentiality is breached, or if cloud providers move regulated data across national borders [33].

1.1 Our Contribution

The contribution of this paper is two-fold. First, we present a list of core security requirements and challenges that must be considered when migrating to a cloud environment. These security requirements were derived based on our experience with migrating existing applications to a private Infrastructure-as-a-Service (IaaS) cloud [30]. We extend this guide by discussing important attack vector characteristics for cloud environments that will pave the way for providing tighter security when building cloud services. Second, in order to tackle the critical cloud security challenges we present PaaSword, an envisaged framework that will maximize and fortify the trust of individual, professional and corporate users to cloud services and applications. PaaSword achieves that by providing storage protection mechanisms, which improves confidentiality and integrity protection of users' data in the cloud while it does not affect the data access functionality.

1.2 Organization

The rest of this paper is organized as follows. In Section 2, we further elaborate on the main data security challenges in cloud-enabled services and applications. In Section 3, we introduce a holistic, data privacy and security by design, framework enhanced by sophisticated context-aware access models and robust policy enforcement and governance mechanisms, aimed at facilitating the implementation of secure and transparent cloud-based applications. In Section 4, we briefly discuss relevant work while in Section 5, we conclude the paper by presenting the next steps for the implementation and evaluation of the proposed framework.

2 Data Security Challenges in the Cloud

According to the Cloud Security Alliance [1], several top security identified threats refer to information disclosure and repudiation, rendering data security as realized through data protection, privacy, confidentiality, and integrity as top priorities. More precisely, the top four threats identified are: data leakage, data loss, account hijacking and insecure APIs. The externalized aspect of outsourcing can make it harder to maintain data integrity and privacy [19] and organizations should include mechanisms to mitigate security risks introduced by virtualization. Especially when they deal with sensitive data, such as health records, the protection of stored information comes as a top priority. Therefore, data security can be seen as the foundation upon which the entire transition to a cloud architecture should be based. Multiple risks must be addressed in order for an organization to guarantee the safety of users' records. One of the most important aspects is security of sensitive information. To this end, the deployment must ensure that all sensitive data is stored in encrypted form. Complementary to this, proper key management must ensure that encryption keys are not revealed to malicious users.

Based on this, it becomes evident that the most critical part of a modern cloud application is the data persistency layer and the database itself. As all sensitive information (including user credentials, credit card info, personal data, corporate data, etc.) are stored in these architectural parts, the database-takeover is the ultimate goal for every adversary.

The Open Web Application Security Project[1] foundation has categorized the database-related attacks (SQL injection) as the most critical ones. The importance of this attack vector is also reflected by respective incident reports. According to the Web Hacking Incidents Database,[2] SQL injections represents 17% of all security breaches examined. These injections were responsible for 83% of the total records stolen, in successful hacking-related data breaches from 2005 to 2011. The criticality of the persistency layer is therefore evident. Most of the security fences that are configured in a corporate environment target the fortification of the so-called network perimeter (e.g. routers, hosts and virtual machines). Although existing intrusion detection systems (IDS) and intrusion prevention systems (IPS), try to cope with database-takeover security aspects (like Snort), the fact that, on the one side, automated exploitation tools (e.g. SQLMap) are widely spread, and, on the other side, IPS and IDS evasion techniques have become extremely sophisticated, denote that the risk of database compromise is greatest than ever. Moreover, by using mechanisms that rely on Web Application Firewalls (WAF) an organization can prevent various types of attacks but it is inadequate to protect against today's sophisticated SQL Injection and DoS attacks [29]. Additionally, internal adversaries in terms of cloud vendors or even unknown vulnerabilities of software platforms and security components widely adopted in cloud-based development may provide malicious access to personal and sensitive data. A recent example was the Heartbleed flaw[3] that constituted a serious fault in the OpenSSL cryptography library, which remained unnoticed for more than two years and affected over 60% of Web servers worldwide. Additionally, regarding the post-exploitation phase, things are even worse in the case where a symmetric encryption algorithm has been employed to protect the application data. The already available cracking toolkits that utilize GPU processing power (e.g. oclHashcat) are able to crack ciphers using brute-force techniques with an attack rate of 162 billion attempts per second.

While most of the attack vectors are exposed in any Software-as-a-Service application by the system administrator's misconfigurations, the database takeover and the post-exploitation of acquired data is under the sole responsibility of the application developer. The application developer is the one responsible both for sanitizing all HTTP-input parameters that could be used as attack vectors, and for reassuring that compromised data will be useless under the existing brute-forcing and reversing techniques. Nevertheless, even if the application developer follows strict guidelines, the mere utilization of an IaaS provider in order to host a Virtual Machine, or for a Platform-as-a-Service (PaaS) provider in order to develop a cloud application, may by itself spawn a multitude of inherent vulnerabilities. These vulnerabilities cannot be tackled effectively as they typically exceed the responsibilities of an application developer.

3 PaaSword Framework

In this section, we present PaaSword, a framework that allows cloud services to maintain a fully distributed and encrypted data persistence layer. The main aim of PaaSword is to help cloud service providers to foster data protection, integrity and confidentiality in the presence of malicious adversaries. To this end, we describe the need for a context-aware security model which serves as the basis of a fine-grained access control scheme, one which allows the per-user management of access rights. In addition to that, we describe a physical distribution, encryption and query middleware that is based on a searchable encryption (SE) scheme [11] which allows legitimate users to directly search over encrypted data, thus ensuring the confidentiality and integrity of the stored data.

3.1 Context-Aware Access Model

PaaSword builds upon a XACML-based[4] context-aware access model, which is needed by the developers to annotate the Data Access Objects of their applications.

[1] https://www.owasp.org/

[2] http://projects.webappsec.org/w/page/13246995/Web-Hacking-Incident-Database

[3] http://www.infosecurity-magazine.com/news/heartbleed-101/

[4] OASIS eXtensible Access Control Markup Language (XACML). https://www.oasis-open.org/

This context model conceptualizes the aspects, which must be considered during the selection of a data-access policy. These aspects may be any kind of information which is machine-parsable [9]; indicatively they may include the user's IP address and location, the type of device that she is using in order to interact with the application as well as her position in the company. These aspects can be interpreted in different ways during the security policy enforcement. In particular, the context aware access model determines which data is accessible under which circumstances by an already-authenticated user.

Access control models are responsible for deciding if a user has the right to execute a certain operation on a specific object. Objects can be a server, an application, an entire database or even a single field in a table row. The user is considered as the *active element* and is called subject. A permission associates an object with an operation (e.g. read, write etc.). Access control models provide a list of permissions that each subject has on certain objects.

Commonly used access control models are the Mandatory Access Control (MAC), the Discretionary Access Control (DAC) and the Role-Based Access Control (RBAC) [12]. Related work discussion, in Section 4, reveals that there are also different approaches, known as Context- Aware Access Control models (CAAC) that are mainly RBAC-based approaches and either do not cover all relevant contextual elements with a reusable security related context model, or are proven hard to maintain in dynamic environments where users often switch roles [3, 41]. In our approach, the process of granting/denying access will be based on dynamically changing parameters, thus our proposed model will rely on a more recent paradigm called Attribute-Based Access Control (ABAC). The context parameters are unique for every single user, so for granting access it is necessary to consider all information associated with a single user. Our approach goes a step further by considering "associated" attributes, thus making access control decisions based on contextual information. ABAC considers a number of attributes that may drive an access decision, which is much needed in the dynamic environment of cloud domain. Nevertheless, ABAC rules that may involve such attributes and the acquired attribute values when evaluating an access request must be exactly matched. For example, deploying the following ABAC rule: 'If the requestor is located inside the Company A, then permit access to Sensitive Data A', implies that during run-time the access control mechanism will be able to detect that the requestor's location is Company A. But, if this location attribute value is 'Room A' (situated inside the same building) after all, it will not lead to an access permission. For this purpose, we introduce the Context-Aware Access Model that is able to capture semantic knowledge, thus coping with such issues by inferring new knowledge when it is needed (e.g. class/sub-class inferencing).

To implement the dynamic change of context parameters in a static access control model, we will use the, so-called, context switches. Depending on the current context, a permission can be granted or denied (switched). This could switch dynamically with every change of the context. Context switches are responsible for managing operational permissions and object permissions. An operational permission gives the right to a subject to perform a specific operation while an object permission gives the right to perform an operation on a specific object.

3.2 Policies Access and Enforcement

Another important aspect of our proposed framework is a middleware that encapsulates capabilities for maintaining the access policies model, annotating and managing data access object annotations, controlling their validity, dynamically interpreting them into policy enforcement rules and for enforcing them. More precisely this middleware provides the following:

a. A transparent key usage for efficient authentication purposes, related to authenticating the origin of the incoming access requests;
b. Annotation capabilities in the form of a tool (can also involve an IDE plug-in) for allowing developers to declaratively create the minimum amount of rule-set that is needed for security enforcement purposes;
c. The dynamic interpretation of the data access object annotations into policy enforcement rules;
d. The governance and quality control of the annotations and their respective policy rules;
e. The formulation and implementation of the overall policy enforcement business logic;

In terms of this middleware, we also consider the reuse and proper extension of technologies for developing an appropriate key management mechanism. This mechanism is necessary for the authentication of different parties that will be involved in the encryption and decryption of data. We aim at constituting the key-usage, transparent to the application usage. This involves the key propagation upon authentication of the user, directly to the security enforcement middleware. For efficiency, we employ a hybrid encryption capitalizing upon the utilization of two different encryption functions. The inner layer will be encrypted with an algorithm that uses a symmetric encryption key K, while the outer layer will use an asymmetric encryption in order to encrypt the symmetric key K. Symmetric encryption allows more efficient schemes compared to the asymmetric encryption. However, combining both techniques help to optimize the efficiency of the underlying protocols without sacrificing security. To this end, PaaSword will rely on both symmetric and asymmetric encryption in order to securely distribute K between legitimate users.

Additionally, we also employ methods and mechanisms for governance and validity control of the data object annotations. More specifically, we focus on the application of an ontology-driven governance approach for: *(i)* the basic management of data object annotations (i.e. storage, retrieval, deletion, etc.), *(ii)* validity checking of the data object annotations (e.g. rejecting any contradicting annotations made by the developer) and *(iii)* dependency tracking among data objects annotations.

Another critical aspect of this middleware is the annotations interpretation mechanism. PaaSword uses such a mechanism for dynamically generating access control policies, during application runtime, based on the interpretation of data object annotations. By doing this, we manage to implement the essential decoupling between the access decisions and the points of use (i.e. Policy Enforcement Points (PEP) of the XACML specification). This interpretation is based on an XACML compliant context model and it can augment the offered functionality of any PaaS provider, with a security-as-a-service layer. To do so, we will use the OASIS XACML as it supports and encourages the separation of the access decision from the point of use.

3.3 Threat Model, Secure Storage & Query Middleware

In this sub-section, we describe the threat model that we consider as well as a brief description of the design of the secure storage that is provided by PaaSword. Furthermore, we also describe the query middleware – a component that is responsible for processing requests regarding the stored encrypted data.

Threat Model Similar to existing works in the area [34, 37], we assume a *semi-honest* cloud provider. In the semi-honest adversarial model, a malicious cloud provider correctly follows the protocol specification. However, she can intercept all messages and may attempt to use them in order to learn information that otherwise should remain private. Semi-honest adversaries are also called *honest-but-curious*.

Furthermore, for the rest of the participants in the protocol we share the threat model with [37], which is based on the Dolev-Yao adversarial model [10] and further assumes that privileged access rights can be used by a remote adversary $\mathcal{ADV}$ to leak confidential information. The adversary, e.g. a corrupted system administrator, can obtain remote access to any host maintained by the provider. However, the adversary cannot access the volatile memory of any guest virtual machine (VM) residing on the compute hosts of the provider. This property is based on the closed-box execution environment for guest VMs, as outlined in Terra [14] and further developed in [42].

Secure Storage A basic tenet of PaaSword is that sensitive data stored on untrusted servers must be always encrypted. This effectively reduces the privacy and security risks since it relies on the semantic security of the underlying cryptosystem, rendering the system relatively immune to internal and external attacks. Having this in mind, we propose a forward-looking design for a cryptographic cloud storage that is based on a symmetric searchable encryption (SSE) scheme similar to the one proposed in [20] and the one used in [27]. In addition to that, we plan to extend the previous work Cumulus4j [18] and MimoSecco [13] in such a way that will fit the needs of the aforementioned SSE scheme. Cumulus4j and MimoSecco presented a SSE scheme that was based on the IND-ICP security notion

[2] that hides relations between different data values of a data row and creates the base for secure database outsourcing.

An SSE scheme allows a user to search over encrypted data without learning any information about the plaintext data. Let $\mathcal{DB} = \{m_1, \ldots, m_n\}$ be a set of n messages (w.l.o.g $\mathcal{DB}$ can be considered as a database). For each $m_i \in \mathcal{DB}$ we extract a set of keywords which can later be used for executing queries. This set of keywords is denoted as $\mathcal{W} = \{w_1, \ldots, w_n\}$. For each $w_i \in \mathcal{W}$ we calculate $H(w_i)$, where $H(\cdot)$ is a cryptographically secure hash function under a secret key K'. Then, we encrypt the elements of $\mathcal{DB}$ with a secret key $\mathsf{K}'' \neq \mathsf{K}'$. By doing this, we create a searchable encrypted index I where each index entry, points to an encrypted list of rows that have a certain keyword. The client can use a trapdoor function to search the index and determine whether a specific keyword is contained in the index.

While the above-mentioned scheme is implemented in previous works [13, 18] it has a limitation that we cover in our proposed framework. More precisely, the current scheme follows a single write/single read (S/S) architecture, which makes it unrealistic for our cloud scenario. To overcome this limitation, PaaSword is using a proxy re-encrpytion component that supports multi write/multi read (M/M) meaning that a group of users based on access rights can both read and write on the encrypted data. To this end, PaaSword involves a key distribution algorithm that extends S/S architecture to M/M. Additionally, a user revocation function is supported by the framework in order to exclude a user, which either acts maliciously or has no longer access rights. This is a crucial and challenging procedure, if we consider that many of the existing SSE schemes [2] do not support user revocation and thus are susceptible to many attacks (Fig. 1).

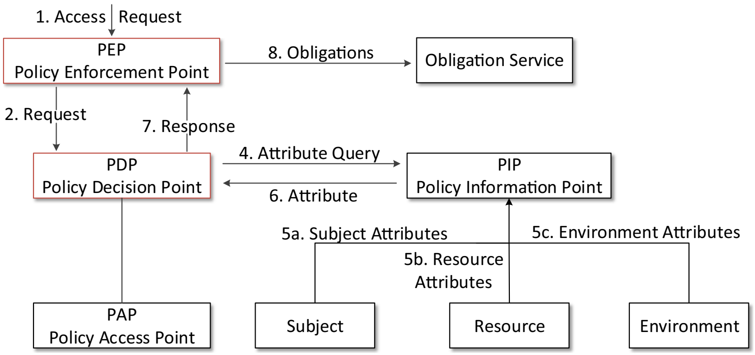
Query Middleware In order to successfully support the SSE scheme described above, PaaSword is using a persistency layer, called Virtual Database (VB) (Fig. 2). VB acts as the intermediary that secures client data before it gets uploaded to the cloud. Additionally, this layer is responsible for processing user queries. In our framework, the VB plays the role of a "trusted third party". Consider, for example, the scenario where a user wants to search for a certain data in PaaSword secured databases. To do so, she will generate a query (q) containing a set of keywords that she is interested in and will send the request to the VB. Upon reception, the VB extracts the keywords from q calculates their hash values and queries the databases where the keywords w_i are stored. If the queries are successful and the keywords exist in one of the tables, VB will obtain the row from the main table that contains the encrypted original data. Upon reception, VB will reply to the user's request by sending the acquired data.

3.4 Overview of the PaaSword Architecture

3.4.1 Conceptual Architecture

The PaaSword compliant cloud applications inherits a fully physical distributed and totally encrypted data persistence layer, which is able to determine on an ad-hoc basis whether an incoming data querying and processing request should be granted access to the target data during application runtime. The transformation process of a traditional application utilizing the PaaSword framework and the way the transformed application secures and protects the users' sensitive data is presented in Fig. 2, which at the same time reveals high level architectural details of the framework.

Fig. 1 High level view of XACML components



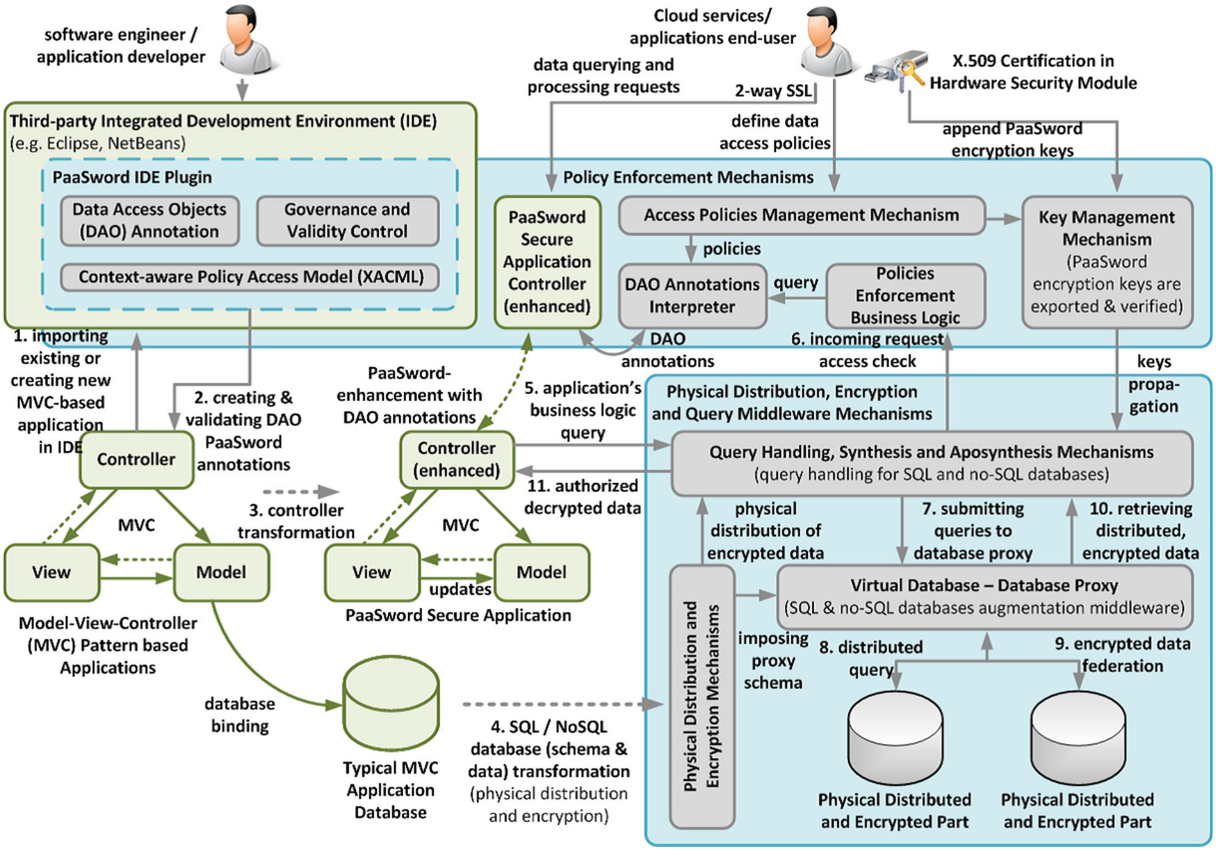

software engineer / application developer
Cloud services/ applications end-user
data querying and processing requests
2-way SSL
X.509 Certification in Hardware Security Module
define data access policies
append PaaSword encryption keys
Third-party Integrated Development Environment (IDE) (e.g. Eclipse, NetBeans)
PaaSword IDE Plugin
Data Access Objects (DAO) Annotation
Governance and Validity Control
Context-aware Policy Access Model (XACML)
Policy Enforcement Mechanisms
PaaSword Secure Application Controller (enhanced)
Access Policies Management Mechanism
policies
DAO Annotations Interpreter
query
Policies Enforcement Business Logic
Key Management Mechanism (PaaSword encryption keys are exported & verified)
DAO annotations
6. incoming request access check
keys propa-gation
1. importing existing or creating new MVC-based application in IDE
2. creating & validating DAO PaaSword annotations
PaaSword-enhancement with DAO annotations
5. application's business logic query
Physical Distribution, Encryption and Query Middleware Mechanisms
Controller
Controller (enhanced)
Query Handling, Synthesis and Aposynthesis Mechanisms (query handling for SQL and no-SQL databases)
11. authorized decrypted data
3. controller transformation
MVC
MVC
physical distribution of encrypted data
7. submitting queries to database proxy
10. retrieving distributed, encrypted data
View
Model
View
Model
updates
PaaSword Secure Application
Virtual Database – Database Proxy (SQL & no-SQL databases augmentation middleware)
Model-View-Controller (MVC) Pattern based Applications
Physical Distribution and Encryption Mechanisms
imposing proxy schema
8. distributed query
9. encrypted data federation
database binding
Typical MVC Application Database
4. SQL / NoSQL database (schema & data) transformation (physical distribution and encryption)
Physical Distributed and Encrypted Part
Physical Distributed and Encrypted Part

Fig. 2 PaaSword framework conceptual architecture

In this framework, we consider applications that adopt and respect the Model-View-Controller (MVC) development pattern [24]. As seen in Fig. 2 (step 1) the application developer imports an existing or creates a new MVC-based application in her favorite integrated development environment (IDE) for which an IDE-specific plug-in will be provided. During the second step of this process the application developer creates annotations at the DAO of the Controller referring to sensitive data that should be protected, according to the XACML-based model and defines the physical distribution, encryption and access rights scheme for each data object. In the third step, the DAO annotations will be checked for their validity and compiled with the overall application code. This will allow the transformation of the application's controller that has been enhanced with XACML-based DAO annotations, leading to the implementation of a PaaSword secure application. In the fourth step, the persistence layer of the application will be physically distributed and encrypted at the schema and instance level according to the incorporated DAO annotations, imposing the schema and driving query handling capabilities of the VB that augments the actual data persistence layer of the application. At application runtime (step 5), each query and processing request of the end-user is forwarded by the enhanced controller to the query handling mechanism that is responsible for the database proxy queries synthesis and aposynthesis. In step 6 and before the submission of the enhanced query to the VB, the query handling mechanism consults the policy enforcement mechanism to determine whether the incoming request should be granted or not. Upon policies enforcement and access permission, the query handling mechanism submits (step 7) the enhanced query to the augmented persistence layer (virtual database). The database proxy that is aware of the physical distribution scheme of the actual application database realizes the distributed query to the physically distributed and encrypted parts of the actual application database (step 8). Next, the federation of the respective encrypted data from the distributed parts of the database takes place (step 9). The federated data synthesis and ad-hoc decryption utilizing the key of the end-user that is transparent to the application and is propagated to the query handling mechanism (step 10). Last, the query handling mechanism delivers the decrypted data to the application controller that forwards them to the end-user through the "view" component of the application.

According to the conceptual view (Fig. 2), each end-user is equipped with a Hardware Security Module, such as USB stick or a smart-phone with digital rights management module, which contains a digital certificate (e.g. X.509). Part of the certificate includes keys that can be exported by the PaaS/IaaS provider. These keys upon exportation and verification are transparently handed over to the query middleware which is responsible for interacting with the VB to encrypt and decrypt the corresponding data.

3.4.2 Conceptual Walkthrough of the Framework

A brief analysis of the basic background concepts that relate to the envisaged security and privacy-by-design framework are discussed in this section. This analysis is a conceptual walkthrough of the framework's usage and complements the bird's eye view of the PaaSword's conceptual architecture described in the previous section. The walkthrough is depicted on Fig. 3.

In order to better describe the conceptual walkthrough of the PaaSword framework, we first identify the main stakeholders:

- *PaaSword Administrator* is authorized to manage the centralized architectural artifacts (e.g. the Context Model);
- *PaaSword Developer* has the option to extend the PaaSword libraries;
- *Product Manager* is able to provide tangible technical requirements to its development team regarding security and privacy issues;
- *Cloud Application Developer* uses the PaaSword libraries during the development process in order to enforce specific type of encryption policies or access policies directly in her code according to the requirements given by the product manager;

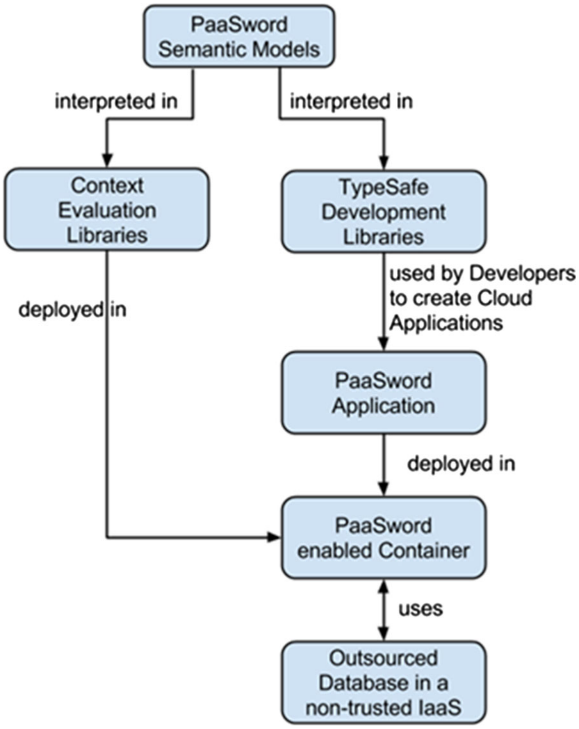

PaaSword Semantic Models
interpreted in
interpreted in
Context Evaluation Libraries
TypeSafe Development Libraries
used by Developers to create Cloud Applications
deployed in
PaaSword Application
deployed in
PaaSword enabled Container
uses
Outsourced Database in a non-trusted IaaS

Fig. 3 Conceptual walkthrough of the PaaSword framework

- *DevOp* is responsible for performing the deployment and the management of a PaaSword-enabled application in a PaaSword-enabled container;
- *Cloud Application Owner* is monetizing the PaaSword enabled application;
- *PaaS Provider* operates the execution container of the PaaSword-enabled application;
- *IaaS Provider* is used in order to host the operational environment of a database;
- *Cloud Application User* interacts with a PaaSword-enabled application;

As depicted in Fig. 3, the root element refers to the PaaSword Semantic Models. The PaaSword Semantic Models refer to a set of ontological models that aim to conceptualize two artifacts:

a. *Transparent encryption/decryption policies* that can be used during run-time by an application in order to protect specific columns in a database. These policies refer to the various database-security mechanisms. They are configured once during the development process by the Product

Manager and the Cloud Application Developer of a PaaSword-enabled application;

b. *Context-driven security policies* that can be applied in the web-endpoints of a PaaSword-enabled application. These policies can be managed even during the run-time in order to permit or refuse access to a user for specific web-endpoints;

Both these modes become usable when they are transformed to Typesafe Development Libraries. Based on their usage, there are two types of libraries:

a. *Development-oriented libraries* that are used by cloud application developers of PaaSword-enabled applications;
b. *Run-time libraries* that perform mainly two tasks; context evaluation of a user's request and transparent encryption/decryption;

At this point it should be clarified that the PaaSword architecture is not bound to a specific programming language or framework. However, the envisioned architecture raises some implementation requirements such as:

- The ability of a programming language to support annotations (or any other metadata framework) which is an essential prerequisite of the PaaSword architecture since transparent encryption/decryption policies and web-endpoints that are controlled by PaaSword policies are defined using annotations;
- The ability of an execution container to support dynamic class-loading which is also an essential prerequisite of the PaaSword architecture since various context evaluation libraries can be dynamically provided to an execution container;

There are many programming languages that support these features; yet the entire set of the use cases that will be supported are JAVA oriented. Therefore, the reference implementation will be JAVA oriented. As a result, from now on the term programming language annotation will refer to a JSR-175[5] JAVA annotation, the term library will refer to a JAVA library and the term execution container will refer to a Java EE execution container. After this clarification, it should be noted that the aforementioned run-time libraries are deployed in a normal execution container in order to be used during run-time. On the other hand, the Typesafe development libraries are used by developers. An example of the typesafe library usage is provided in the following code-listing (Algorithm 1).

Algorithm 1 Indicative usage of the Typesafe development libraries

```
@Entity
public class User {
    @Id
    @GeneratedValue
    private Long id ;

    @Column( nullable=false )
    @PaaSwordProtected ( Algorithm=Symmetric
       .AES _CBC_NoPadding,
    TDEPolicy=Policy. Monolithic )
    private String creditcard ;
```

According to the example listed in Algorithm 1, a developer annotated one column of its database (a relational database has been used in this case) using one JAVA annotation named PaaSwordProtected. This annotation can be used and validated only if a specific library that contains the metadata definition exists in the development environment. This annotation shall be interpreted by the execution container in order to treat the annotated column in a specific way. More precisely, the developer expects that all entries in the specific column will be symmetrically encrypted using a specified algorithm and a specific transparent encryption policy.

However, the expected behavior of the developer is realized by respective run-time libraries that implement the actual container-logic. Therefore, for each annotation there will be specific handlers that implement the actual logic. The common denominator between the annotations and the run-time handlers is the PaaSword context model. Beyond exposing the functional capabilities of PaaSword to the developers, development libraries serve the purpose of type validation, i.e. they guarantee that a specific annotation is used in a proper element (e.g. @Column in our case), the annotation is configured correctly (e.g. Algorithm and TDE Policy are provided in our case) and finally proper arguments are provided per parameter (e.g. AES_CBC_NoPadding in our case).

[5] https://jcp.org/en/jsr/detail?id=175

On the other hand, the PaaSword enabled Container is responsible to load and orchestrate correctly the libraries that interpret the annotations that are used by the developers. This interpretation may refer to performing transparent data encryption/decryption in an outsourced database (see Fig. 3), and/or to performing context-based policy enforcement.

3.4.3 PaaSword's High Level Components

In this section, we provide a coarse grained view of the core architectural components that comprise the PaaSword reference architecture, which relies on the concepts that have been discussed above.

A security and privacy-by-design framework involves many stakeholders as described above. Each of these stakeholders relies on different components that complement each other in order to implement the conceptual walkthrough present on Fig. 3. The following figure (Fig. 4), presents a coarse grained view of the PaaSword components. As depicted, these components are grouped in various zones according to the role that they possess regarding the PaaSword conceptual workflow.

Starting from the zone of the PaaSword Central Administration (PCA), it consists of the following components:

- *Semantic Model Management (SMM):* It is the component that manages the PaaSword Context Model. This model, is a multi-faceted and multi-purpose model. On the one hand, the model aims to define the functional aspects of the PaaSword framework that can be directly used by the developers and on the other hand it conceptualizes parametric contextual information (e.g. Location, Time of Interaction etc.) of a request that can be used in order to perform policy enforcement. The SMM offers an interface for the creation and editing of the model as well as a structural and business validation of the model entities. Moreover, some facets of the model are created centrally and

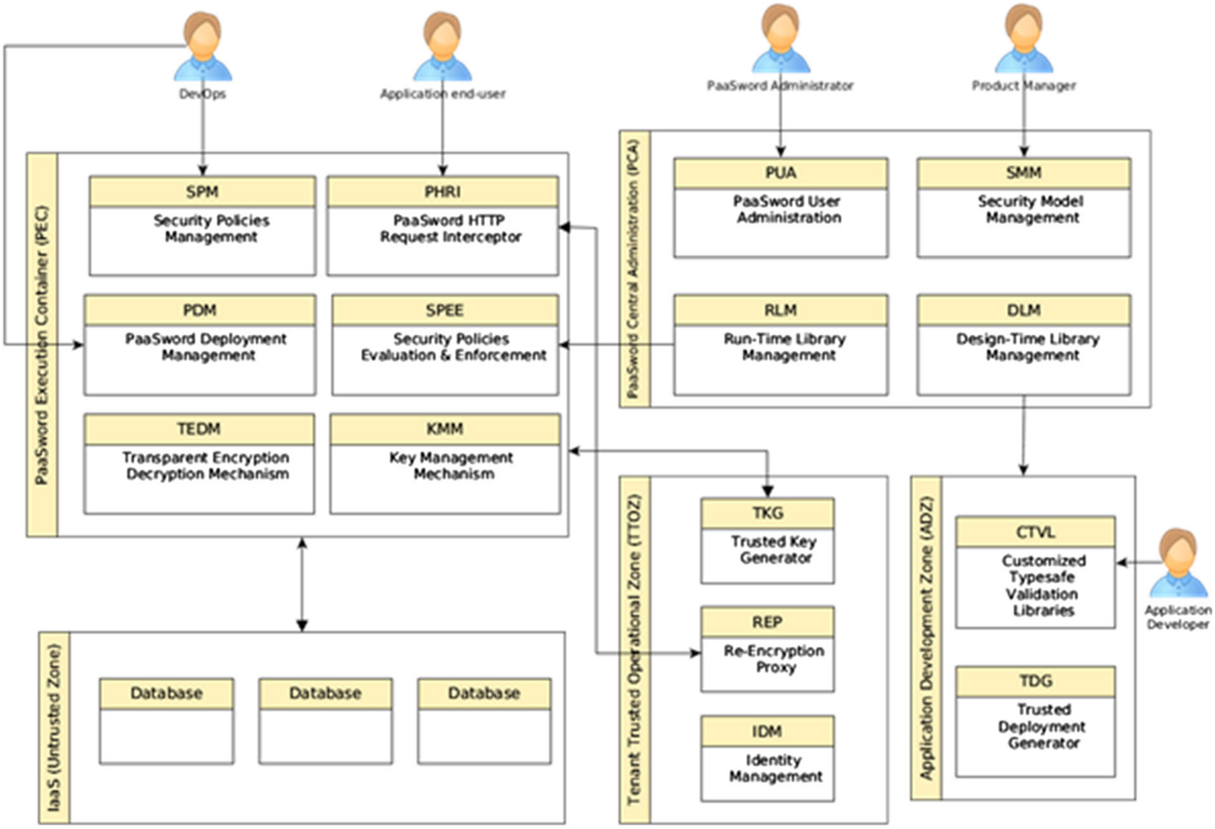

DevOps
Application end-user
PaaSword Administrator
Product Manager
PaaSword Execution Container (PEC)
SPM
Security Policies Management
PHRI
PaaSword HTTP Request Interceptor
PDM
PaaSword Deployment Management
SPEE
Security Policies Evaluation & Enforcement
TEDM
Transparent Encryption Decryption Mechanism
KMM
Key Management Mechanism
PaaSword Central Administration (PCA)
PUA
PaaSword User Administration
SMM
Security Model Management
RLM
Run-Time Library Management
DLM
Design-Time Library Management
Tenant Trusted Operational Zone (TTOZ)
TKG
Trusted Key Generator
REP
Re-Encryption Proxy
IDM
Identity Management
Application Development Zone (ADZ)
CTVL
Customized Typesafe Validation Libraries
TDG
Trusted Deployment Generator
Application Developer
IaaS (Untrusted Zone)
Database
Database
Database

Fig. 4 PaaSword Framework's components

maintained while some other parts are adapted to the customer's need. That is the reason why on Fig. 4 the Product Manager interacts directly with the SMM.

- *Design-Time Library Management (DLM):* It is the component that generates a metadata annotation library which is bound to the PaaSword Context Model. As discussed earlier, this library is used by developers to annotate specific part of their code based on the security-related functionality that they require. In the frame of the project these libraries rely on JSR-175 standard since the annotations will be used on JAVA projects. Although the Context Model contains both centrally managed concepts (e.g. supported encryption/decryption algorithms) and product-specific concepts (e.g. available roles of a product) the library that is generated by the DLM is a unified one that covers both categories of concepts. Finally, it should be highlighted that the scope of the DLM generated library is to make the procedure of PaaSword adoption error-free since the library usage guarantees that a specific annotation is used in a proper element and it is configured and initialized correctly (see Algorithm 1).
- *Run-Time Library Management (RLM):* It is the component that manages the run-time libraries that are bound to the PaaSword context model. These libraries are used during the run-time, upon their deployment in an execution container in order to provide the functionality that is driven by the source code annotations. It should be clarified that while the DLM provides an automatic interpretation of the PaaSword Context Model to a development library, the run-time libraries are manually registered and tested by PaaSword-certified engineers through an appropriate editor. These libraries provide the guarantees that the PaaSword-specific annotations that accompany the code are properly handled by a PaaSword container. They also contain a strict exception handling and test-assertion layer since these libraries can be deployed, upgraded or undeployed dynamically from an execution container.
- *PaaSword User Administration (PUA):* It is the component that manages the registration and the lifecycle of the PaaSword users. According to the PaaSword framework's walkthrough each *Independent Software Vendor* (ISV) that wishes to adopt PaaSword should register in order to be able to *(a)* manage the user-defined aspects of the PaaSword Context Model and *(b)* gain access to the development libraries that are automatically generated.

Following the conceptual walkthrough that was introduced in Fig. 3, the next zone is the Application Development Zone (ADZ). Each ISV that wishes to adopt the PaaSword framework should register to PCA in order to gain access to the PaaSword exported libraries. PaaSword libraries can be used by the developers of ISVs to create PaaSword enabled applications. The following components belong to this zone:

- *Customized Typesafe Validation Libraries (CTVL):* During the developer's guidance for defining effective, meaningful and mutually-consistent security controls through annotations, their compliance against a set of security-related business rules should be checked. These rules, hereafter referred to as the *annotation-formation* (AF) rules, are essentially policies which specify the allowable ingredients of these web-endpoint annotations and hence facilitate their creation. The CTVL essentially implements these AF rules.
- *Trusted Deployment Generator (TDG):* It is the component that configures a PaaSword enabled application in order to be deployed to an execution container. We refer to the final ready-to-be deployed application as deployment archive. The deployment archive entails a specific format which will be validated during the deployment process (a.k.a. bootstrapping). TDG component is used to: *(a)* inject the deployment archive with the proper digital certificates that may be needed (it depends on the encryption/decryption policy); *(b)* inject proper configuration files that will be used during the run-time (e.g. the official URI of the PaaSword run-time libraries) and *(c)* digitally sign the final deployment archive since during bootstrapping a certificate-based verification process will be performed.

The next zone is the PaaSword Execution Container which practically encapsulates all the run-time components of the PaaSword framework. Any execution container can be upgraded to a PaaSword enabled container as if it is equipped with the PaaSword run-time libraries. Although PaaSword framework can be applied to several types of execution containers; yet for the sake of the reference implementation only JAVA EE containers will be elaborated. The PaaSword Execution Container is able to interpret the annotations of the PaaSword applications during run-time. Its main components include:

- *PaaSword Deployment Management (PDM):* It is the component that *(a)* is responsible to validate the deployment archive that is submitted by a DevOp; *(b)* validates the annotations of the application by introspecting the deployment assembly *(c)* validates the existing operational environment by checking that each annotation can be interpreted by a specific run-time library and *(d)* performs all the appropriate steps that are required in order for the application to become operational (e.g. initializes the database).
- *Transparent Encryption & Decryption Mechanism (TEDM):* It is one of the most crucial components of the PaaSword reference architecture. This component is responsible to perform the transparent encryption/decryption tasks so as to protect the data that reside on the (fully untrusted) database. The transparent encryption mechanism is bound to specific policies that will be supported. A policy defines whether the structure of the database schema will be altered or not, the type of the encryption algorithm and the key management methodology.
- *Key Management Mechanism (KMM):* It is the component that performs key management operations that may be required by the TEDM component. More specifically, one of the supported encryption/decryption policies imposes the usage of a key which is created based on the key generation algorithm and is delivered to the end-user. The key generation algorithm is performed by a respective component (TKG, see below) which belongs to a fully trusted zone (operated under the full supervision of the tenant).
- *Security Policy Evaluation and Enforcement (SPEE) & Security Policy Management (SPM):* These are two complementary components that are responsible to handle the policies that are defined by the code-annotations and can be edited by the DevOp during run-time. On the one hand, the SPEE component is responsible to perform the context evaluation of a Cloud Application User (making use of libraries that are generated by the RLM component that was discussed above) and the policy enforcement. Policy enforcement will follow the XACML metamodel. On the other hand, the SPM is responsible to alter the instances of the context model that is used for policy enforcement. For example, if a web end-point is controlled by the end user's location, then a DevOp can provide (or remove) instances of locations that will affect the policy enforcement.
- *PaaSword Request Interceptor (PHRI):* It is the component that is responsible to forward the HTTP traffic of an end-user to a specific proxy, which is addressed as the Re-Encryption Proxy, and operates under the supervision of the tenant. This component is used only in the case of searchable encryption policy is applied.

Finally, the architecture is complemented by some components that belong to the *Tenant Trusted Operational Zone* (TTOZ). This zone operates under the supervision of the PaaSword applications' tenant and therefore it is considered trusted. The main components in this zone include:

- *Trusted Key Generator (TKG):* As already discussed, it is the component that is responsible for the generation and management of encryption/decryption keys which is used only in the case of the shared key policies.
- *Re-Encryption Proxy (REP):* It is the component that processes an end user's interaction with the application in order to perform the encryption/decryption process outside of the PEC container. This component is only usable in the case of the searchable encryption policy.
- *Identity Management (IDM):* It is the component that manages the authentication of all the interacting stakeholders.

4 Related Work

Among the most significant security related concerns in dynamic and heterogeneous environments especially in cloud-enabled systems is the access control that should be able to consider most of the dynamic aspects of such environments. The emerging and ubiquitous computing environments need security control that is easily adaptable to the changing user or environmental contexts. Context information used in an access control decision can be defined as any relevant information about the state of a relevant contextual entity or the state of any relevant relationship between different relevant entities at a particular time that should be taken into account before granting or rejecting a specific access request. From this perspective, context-awareness relates to the use of this context information for access control decision making. In the literature, there are three basic access control models [8], namely Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-Based Access Control (RBAC). All these models are known as identity based access control models where user (subjects) and resources (objects) are identified by unique names [22]. Most of the traditional security models are context insensitive. Nevertheless, a fourth type has been recently identified, the Attribute Based Access Control (ABAC), as an attempt to cope more efficiently with the context. Several variations of the most current type are discussed below.

In an attempt to reinforce the security of remote service accesses, researchers introduced the concept of location-aware access control (LAAC), which allows a system to grant, or deny, access to users based on their physical location. LAAC models typically extend the three basic access control models DAC, MAC and RBAC [8]. Even though LAAC protocols have been studied extensively [5], there is a clear lack of schemes that determine user access not only on the basis of the users' physical location and credentials, but also on the additional pertinent contextual information. In addition, other approaches like [4] incorporate only specific types of contexts such as location and time. Kulkarni et al. [25] proposed a Context-aware RBAC (CA-RBAC) model for pervasive applications that considers user and resource attributes as context constraints. He et al. [38], considered access control for Web services based on the roles and introduced a CAAC policy model considering the user, resource and environment concepts. Toninelli et al.[39], proposed a CAAC approach which provides resource access permission on the basis of resource availability, user roles, location and time. It involves an ontology-based framework that includes both context and policy models. The disadvantage of the above mentioned approaches is that they only consider specific types of contexts which are not sufficient and generic enough to be used in dynamic cloud environments. Lodderstedt et al. [26], proposed SecureML, an RBAC-based modelling language for integrating access control information into application models expressed in UML. To address the inefficiencies of traditional RBAC models, they also introduced the concept of authorization constraints. These are defined in the Object Constraint Language (OCL) and express preconditions for granting access to one or more operation on particular resources. Although these preconditions take into account the dynamic state of the target resource, the current call, or the environment, they are not based on an extensible and reusable context model, rendering this approach overly static for the requirements of cloud-based systems.

The work reported in [7] was the first to introduce the notion of context-aware access control (CAAC), motivated by applications for intelligent homes. More precisely, the authors introduced a set of services which are enabled based on the location of objects or subjects. The main drawback of the proposed model is the fact that it does not support dynamically generated context, whilst it fails to address important requirements such as multi-granularity of position. Other existing CAAC models are predominantly based on RBAC [21] and typically target a specific domain [6].These models, however, have not been designed to provide fine-grained data access control, e.g. by providing the ability to specify different access rules for different rows of a database. Nevertheless, the necessary separation of concerns, requires a declarative representation of policies, one which is orthogonal to the code of the enforcement mechanisms. Several syntactic descriptions exist (RuleML,[6] XACML,[7] WS-Trust[8]) but they fail to capture the knowledge lurking behind policies: they are merely data models that lack any form of semantic agreement beyond the boundaries of the organisations that developed them. Any interoperability relies on the use of vocabularies that are shared among all parties involved in an interaction. This has a number of limitations: *(i)* it leads to

[6] http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.01

[7] eXtensible Access Control Markup Language (XACML) Version 3.0 – http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[8] WS-Trust 1.3 – http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.doc

ad-hoc reasoning about policy compliance, one which is tied to the specific vocabularies that express the rules according to which the reasoning takes place; *(ii)* it limits the reusability and portability of policies; *(iii)* it precludes the identification of inter-policy relations; *(iv)* it limits the ability to perform policy governance. In order to overcome such limitations, semantically-rich approaches to the specification of policies have been brought to the attention of the research community [17, 32, 40]. These generally embrace Semantic Web representations for capturing what we term action-oriented policies, i.e. policies which control if a particular actor can perform a specified action on, or through the use of, a certain resource. More specifically, these approaches employ ontologies in order to assign meaning to actors, actions and resources. Being "a formal, explicit specification of a shared conceptualization" [16], an ontology provides a flexible, formal, and unambiguous means of agreement upon the semantics of concepts, and their interrelations, in a given domain of discourse. Whilst achieving a proper separation of concerns between policy specification and policy enforcement, the aforementioned semantically-enhanced approaches rely on bespoke, non-standards-based, ontologies for the representation of policies. Although such ontologies may be suitable for characterizing certain action-oriented policies, they generally lack the expressivity for addressing the business details that such policies should support. In addition, their reliance on OWL, despite the obvious benefits stemming from the rich set of properties that OWL offers, raises concerns about the degree to which these approaches are lightweight, hence their performance is questionable.

Regarding the policy management, as shown by a recent survey of methods in contemporary open source registry and repository systems [23], a major weakness is the lack of proper separation of concerns. The policy definition and policy enforcement are entangled in the implementation of a single software component – the policy checker. The rules that a policy comprises are typically encoded in an imperative manner, as part of the same code that checks for potential policy violations. This has a number of negative repercussions among which is the lack of portability and the lack of explicit representation of policy relationships.

The data distribution and encryption algorithms are also important aspects towards trusted cloud services [35] and applications. In [15], C. Gentry presented the first fully homomorphic encryption scheme that enables semantically secure outsourcing to the cloud. The cloud provider operates blindly on the encrypted data and yields the correct, encrypted result. Nevertheless, its practicality is in question as the latest implementations are still orders of magnitude slower than just downloading all encrypted data, decrypting, processing and encrypting it locally and finally uploading it again. In another interesting approach [36], the concept of onions is used. Onions are managed monolithically by a proxy, acting as an adapter between the user and the storage back-end. Each attribute in a relational table is initially asymmetrically encrypted. If certain queries for an attribute are issued, layers of the onion are peeled off, resulting in another, less secure onion. CryptDB uses a novel scheme for order preserving encryption that leaks no information about the data besides order and thus allows sorting encrypted data securely. The main drawback of CryptDB is the lack of security guarantees to the client. More precisely, the only guarantee is that an untrusted server will learn only the information that is necessary to process the query. This may cause every attribute to be reduced to the plain text in the worst case. Also, peeling off layers cannot be reversed, so a single query is sufficient to lower the security forever.

5 Conclusions

In this paper, we proposed the PaaSword framework that can be exposed as a service at the level of PaaS. This framework can tackle the identified cloud security requirements and challenges that should be considered in order to enhance data protection, integrity and confidentiality in the presence of malicious adversaries. The envisaged PaaSword goes beyond the state-of-the-art and allows cloud services to maintain a fully distributed and encrypted data persistence layer. Our framework involves a context-aware security model, the necessary policies enforcement mechanism along with a physical distribution, encryption and query middleware.

Future work involves the implementation of the proposed framework into a fully functional solution which will be validated through the following five pilots in various industrial contexts: *(i)* Encrypted persistency as a service in a PaaS provider, *(ii)* Intergovernmental secure document and personal data exchange, *(iii)* Secure sensors data fusion and analytics, *(iv)* Protection of personal data in a multi-tenant CRM, *(v)* Protection of sensible enterprise information in multi-tenant ERP. These pilots will allow us to test PaaSword and validate its added value in a variety of heterogeneous cases.

Finally, an area that will benefit from PaaSword framework is the so called participatory sensing [28]. The evolution of this field is driven by the introduction of sensors into mobile devices. The openness of such systems and the richness of user data they entail (users can collect valuable data from everywhere) raise significant concerns for their storage and processing. Protocol designers by having PaaSword framework in hands will be able to incorporate secure cloud computing techniques in order to facilitate the storage and processing of the vast amount of collected data.

References

1. Alliance, C.S.: The notorious nine – cloud computing top threats in 2013 (2013)
2. Bösch, C., Hartel, P., Jonker, W., Peter, A.: A survey of provably secure searchable encryption. ACM Comput. Surv. **47**(2), 18:1–18:51 (2014). doi:10.1145/2636328
3. Boustia, N., Mokhtari, A.: Representation and reasoning on orbac: Description logic with defaults and exceptions approach. In: Third International Conference on Availability, Reliability and Security.ARES 08, pp. 1008–1012. doi:10.1109/ARES.2008.144 (2008)
4. Chandran, S.M., Joshi, J.B.D.: Lot-rbac: a location and time-based rbac model. In: Proceedings of the 6th International Conference on Web Information Systems Engineering, pp. 361–375. Springer, Berlin, WISE'05. doi:10.1007/11581062_27 (2005)
5. Cleeff, A.V., Pieters, W., Wieringa, R.: Benefits of location-based access control: A literature study. In: Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Physical and Social Computing, pp 739–746. IEEE Computer Society, Washington, DC, GREENCOM-CPSCOM '10. doi:10.1109/GreenCom-CPSCom.2010.148 (2010)
6. Costabello, L., Villata, S., Gandon, F.: Context-aware access control for rdf graph stores. In: Raedt, L.D., Bessière, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., Lucas, P.J.F. (eds.) ECAI, IOS Press, Frontiers in Artificial Intelligence and Applications, vol 242, pp 282–287. http://dblp.uni-trier.de/db/conf/ecai/ecai2012.html (2012)
7. Covington M.J., Long W., Srinivasan S., Dev A.K., Ahamad M., Abowd G.D.: Securing context-aware applications using environment roles. In: Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies, ACM, New York, NY, USA, SACMAT '01, pp 10–20. doi:10.1145/373256.373258 (2001)
8. Decker, M.: Modelling of location-aware access control rules. In: Handbook of Research on Mobility and Computing: Evolving Technologies and Ubiquitous Impacts, pp. 912–929. IGI Global. doi:10.4018/978-1-60960-042-6.ch057 (2011)
9. Dey, A.K.: Understanding and using context. Pers. Ubiquit. Comput. **5**(1), 4–7 (2001). doi:10.1007/s007790170019
10. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Trans. Inf. Theory **29**(2), 198–208 (1983)
11. Dowsley, R., Michalas, A., Nagel, M.: A report on design and implementation of protected searchable data in iaas. Tech. rep. Swedish Institute of Computer Science (SICS) (2016)
12. Ferrari, E.: Access Control in Data Management Systems. Morgan and Claypool Publishers (2010)
13. Gabel, M., Hübsch, G.: Secure database outsourcing to the cloud using the mimosecco middleware. In: Krcmar, H., Reussner, R., Rumpe, B. (eds.) Trusted Cloud Computing, pp. 187–202. Springer International Publishing, Berlin (2014). doi:10.1007/978-3-319-12718-7_12
14. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: a virtual machine-based platform for trusted computing. In: ACM SIGOPS Operating Systems Review, vol. 37, pp. 193–206 (2003)
15. Gentry C.: A fully homomorphic encryption scheme. PhD thesis, Stanford, CA, USA, aAI3382729 (2009)
16. Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. Int. J. Hum.-Comput. Stud. **43**(5–6), 907–928 (1995). doi:10.1006/ijhc.1995.1081
17. Hu, H., Ahn, G.J., Kulkarni, K.: Ontology-based policy anomaly management for autonomic computing. In: 2011 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), pp. 487–494 (2011)
18. Huber, M., Gabel, M., Schulze, M., Bieber, A.: Cumulus4j: a provably secure database abstraction layer. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L., Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L. (eds.) CD-ARES Workshops, Springer, Lecture Notes in Computer Science, vol. 8128, pp. 180–193. http://dblp.uni-trier.de/db/conf/ares/cd-ares2013w.html (2013)
19. IBM: Security and high availability in cloud computing environments. Tech. rep. IBM SmartCloud Enterprise, East Lansing. http://www-935.ibm.com/services/za/gts/cloud/Security_and_high_availability_in_cloud_computing_environments.pdf (2011)
20. Kamara, S., Lauter, K.: Cryptographic cloud storage. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J., Sako, K., Sebé, F. (eds.) Financial Cryptography and Data Security, Lecture Notes in Computer Science, vol 6054,

pp. 136–149. Springer, Berlin (2010). doi:10.1007/978-3-642-14992-4_13

21. Kayes, A.S.M., Han, J., Colman, A.: An ontology-based approach to context-aware access control for software services. In: Lin, X., Manolopoulos, Y., Srivastava, D., Huang, G. (eds.) WISE (1), Springer, Lecture Notes in Computer Science, vol. 8180, pp. 410–420. http://dblp.uni-trier.de/db/conf/wise/wise2013-1.html (2013)
22. Khan, A.R.: Access control in cloud computing environment. ARPN J. Eng. Appl. Sci. **7**(5), 613–615 (2012)
23. Kourtesis D., Paraskakis I.: A registry and repository system supporting cloud application platform governance. In: Proceedings of the 2011 International Conference on Service-Oriented Computing, pp. 255–256. Springer, Berlin, ICSOC'11. doi:10.1007/978-3-642-31875-7_36 (2012)
24. Krasner, G.E., Pope, S.T.: A cookbook for using the model-view controller user interface paradigm in smalltalk-80. J Object Oriented Program **1**(3), 26–49 (1988). http://dl.acm.org/citation.cfm?id=50757.50759
25. Kulkarni, D., Tripathi, A.: Context-aware role-based access control in pervasive computing systems. In: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, ACM, New York, NY, USA, SACMAT '08, pp 113–122. doi:10.1145/1377836.1377854 (2008)
26. Lodderstedt T., Basin D.A., Doser J.: Secureuml: a uml-based modeling language for model-driven security. In: Proceedings of the 5th International Conference on The Unified Modeling Language, UML '02, pp 426–441. Springer, London. http://dl.acm.org/citation.cfm?id=647246.719477 (2002)
27. Michalas, A., Dowsley, R.: Towards trusted ehealth services in the cloud. In: 1st International Workshop on Cloud Security and Data Privacy by Design (CloudSPD'15), co-located with the 8th IEEE/ACM International Conference on Utility and Cloud Computing (UCC), IEEE/ACM (2015)
28. Michalas, A., Komninos, N.: The lord of the sense: A privacy preserving reputation system for participatory sensing applications. In: Computers and Communication (ISCC), 2014 IEEE Symposium, pp 1–6. IEEE (2014)
29. Michalas, A., Komninos, N., Prasad, N.R., Oleshchuk, V.A.: New client puzzle approach for dos resistance in ad hoc networks. In: 2010 IEEE International Conference Information Theory and Information Security (ICITIS), pp. 568–573. IEEE (2010)
30. Michalas, A., Paladi, N., Gehrmann, C.: Security aspects of e-health systems migration to the cloud. In: 2014 IEEE 16th International Conference on e-Health Networking, Applications and Services (Healthcom), pp 212–218. IEEE (2014)
31. Micro, T.: The need for cloud computing security. In: A Trend Micro White Paper (2010)
32. Nejdl, W., Olmedilla, D., Winslett, M., Zhang, C.C.: Ontology-based policy specification and management. In: Proceedings of the Second European Conference on the Semantic Web: Research and Applications, ESWC'05, pp 290–302. Springer, Berlin. doi:10.1007/11431053_20 (2005)
33. Paladi, N., Michalas, A.: One of our hosts in another country: challenges of data geolocation in cloud storage. In: 2014 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE), pp. 1–6. doi:10.1109/VITAE.2014.6934507 (2014)
34. Paladi, N., Michalas, A., Gehrmann, C.: Domain based storage protection with secure access control for the cloud. In: Proceedings of the 2014 International Workshop on Security in Cloud Computing, ASIACCS '14. ACM, New York. doi:10.1145/2600075.2600082 (2014)
35. Paladi, N., Gehrmann, C., Michalas, A.: Providing user security guarantees in public infrastructure clouds. IEEE Trans. on Cloud Comput. **PP**(99), 1–1 (2016). doi:10.1109/TCC.2016.2525991
36. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: Cryptdb: Protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP '11, pp 85–100. ACM, New York. doi:10.1145/2043556.2043566 (2011)
37. Santos, N., Gummadi, K.P., Rodrigues, R.: Towards trusted cloud computing. In: Proceedings of the 2009 Conference on Hot Topics in Cloud Computing, USENIX, Berkeley, CA, HotCloud'09. http://dl.acm.org/citation.cfm?id=1855533.1855536 (2009)
38. Shen, H., Cheng, Y.: A context-aware semantic-based access control model for mobile web services. In: Shen, G., Huang, X. (eds.) Advanced Research on Computer Science and Information Engineering, Communications in Computer and Information Science, vol 153, pp. 132–139. Springer, Berlin (2011). doi:10.1007/978-3-642-21411-0_21
39. Toninelli, A., Montanari, R., Kagal, L., Lassila, O.: A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In: Proceedings of the 5th International Conference on The Semantic Web, ISWC'06, pp 473–486. Springer, Berlin. doi:10.1007/11926078_34 (2006)
40. Uszok, A., Bradshaw, J.M., Johnson, M., Jeffers, R., Tate, A., Dalton, J., Aitken, S.: Kaos policy management for semantic web services. IEEE Intell. Syst. **19**(4), 32–41 (2004). doi:10.1109/MIS.2004.31
41. Verginadis, Y., Mentzas, G., Veloudis, S., Paraskakis, I.: A survey on context security policies. In: 1st International Workshop on Cloud Security and Data Privacy by Design (CloudSPD'15), co-located with the 8th IEEE/ACM International Conference on Utility and Cloud Computing (UCC), IEEE/ACM (2015)
42. Zhang, F., Chen, J., Chen, H., Zang, B.: Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp 203–216. ACM (2011)