Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

PaaSword: A Data Privacy and Context-aware Security Framework for Developing Secure Cloud Applications

Technical and Scientific Contributions

Efi Papatheocharous[1], Spyros Mantzouratos[3], Panagiotis Gouvas[2], Gunther Schiefer[4], Sebastian T. Schork[5], Mohamed Ahmed Abdelraheem[1], Jeremias Mechler[4], Matthias Gabel[4], George Moldovan[6], Kateryna Yurchenko[4], and Thomas Carnehult[1]

[1] RISE ICT/SICS, Isafjordsgatan 22, Kista, Sweden name.surname@ri.se
[2] Ubitech Ltd., Athens, Greece pgouvas@ubitech.eu
[3] IntraSoftInternational S.A., 2B Rue Nicolas Bov, Luxembourg spyros.mantzouratos@intrasoft-intl.com
[4] Karlsruhe Institute of Technology, Kaiserstr. 12, Karlsruhe, Germany name.surname@kit.edu
[5] CAS Software AG, CAS-Weg 1/5, Karlsruhe, Germany sebastian.schork@cas.de
[6] SIEMENS SRL, Blv. Eroilor 3A, Brasov, Romania george.moldovan@siemens.com

Abstract. Most industries worldwide have entered a period of reaping the benefits and opportunities cloud offers. At the same time, many efforts are made to address engineering challenges for the secure development of cloud systems and software. With the majority of software engineering projects today relying on the cloud, the task to structure end-to-end secure-by-design cloud systems becomes challenging but at the same time mandatory. The PaaSword project has been commissioned to address security and data privacy in a holistic way by proposing a context-aware security-by-design framework to support software developers in constructing secure applications for the cloud. This chapter presents an overview of the PaaSword project results, including the scientific achievements as well as the description of the technical solution. The benefits offered by the framework are validated through two pilot implementations and conclusions are drawn based on the future research challenges which are discussed in a research agenda.

1 Introduction

Many organizations worldwide have already entered a period of reaping the benefits and opportunities that cloud computing offers, including agility, scalability, cost benefits and business growth [1]. Moreover, the worldwide public cloud services market is expected to grow 18% in 2017 and even more the upcoming years [1]. The market figures show that today Small and Medium-sized Enterprises (SMEs) have moved further along the cloud adoption curve and by 2019 more than 30% of the 100 largest vendors will shift their future software investments from “cloud-first” to “cloud-only” investments [1].

These trends of growth naturally led to many international and pan-European efforts (e.g., [2],[3]) addressing engineering-driven activities for the secure development of cloud systems and software. The European Security Research and Innovation Forum

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

(ESRIF) was formed to draw a security research and innovation agenda for the EU. ESRIF claims that security infrastructure, equipment and services may become a market success for research and innovation, only if they can meet wide public acceptance. In their report [2] they define the "security-by-design" concept as the need "*to embed security in the technology and system development from the early stages of conceptualisation and design*". This means that the systems and software development methods, processes and tools will need to allow for the integration of "security-by-design" or "security-by-default" concepts. In [3], the National Institute of Standards and Technology (NIST) describes a set of systems security engineering process extensions for the International Standard *ISO/IEC/IEEE 15288* [4]. The security-related design principles described within the architecture definition process, include, but are not limited to: separation, isolation, encapsulation, non-bypassability, layering modularity, hierarchical trust, hierarchical protection and secure distributed composition [4].

1.1 Motivation

The cloud paradigm however introduces inherent capabilities and limitations, related to the technology used, as shown by several recent incidents [5]. In 2014 multiple security incidents, resulting in 2 million customers' personal information being stolen, were reported by British telecom provider TalkTalk, due to its failure to encrypt customer data. In 2015 a hacker demanded ransom of 15,000 USD from BitDefender, an antivirus firm, for stolen customer usernames and passwords caused by a security vulnerability in its public cloud application hosted on AWS (Amazon Web Services). In 2015, the US Internal Revenue Service (IRS) exposed over 300,000 records via a vulnerable API (Application Programming Interface). In response to such incidents, the CSA (Cloud Security Alliance) listed the following top cloud computing threats [6]:

- **Data breaches:** The incidents in which sensitive or confidential information is released, viewed, stolen or used by an individual who is not authorized to do so.
- **Weak identity, credential and access management:** The incidents which occur and cause security breaches and attacks due to lack of scalable identity access management systems, failure to use multi-factor authentications, weak password identity use with appropriate secure cryptographic keys, passwords and certificates.
- **Insecure APIs:** The security and availability of cloud services depends on the security of the set of software UIs (User Interfaces) and APIs their customers use. These need to be designed to assure security and protection against accidental and malicious authentication and access control.
- **System and application vulnerabilities:** Vulnerabilities and exploitable bugs existing in applications and systems that allow attackers to infiltrate a computer system for stealing data, taking over the system or compromising normal quality system behaviour (e.g., availability, responsiveness).
- **Account hijacking:** Incidents caused by attacks such as phishing, fraud and exploitation of software vulnerabilities, misuse of credentials and passwords.

Specifically addressing the security concerns stemming from migrating enterprise sensitive data and operations to cloud infrastructures is a challenging task for the following primary reasons: (a) there is increased variability in the systems type and purpose,

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

(b) systems span over diverse geographical locations, (c) there is rapid proliferation in type as well as in growing size and complexity of components and technologies, and, (d) there is a growing need in elasticity (resilience) and dynamicity in the system's structure, behavior as well as interactions. Thus, the task to build end-to-end secure-by-design cloud systems becomes challenging but at the same time mandatory.

1.2 Contribution

To this end, the PaaSword[7] project, funded by the EU Horizon 2020 programme, aims at establishing a holistic data privacy and context-aware security-by-design framework to facilitate the wider adoption of enterprise cloud computing. More specifically, the objective for PaaSword, is to address some of the top cloud computing threats, as introduced earlier, i.e., data access breaches, insecure APIs and weak mechanisms for access control and encryption. Fig. 1 introduces the contribution of and motivation for the PaaSword framework direction and implementation. It offers two main functionalities to developers of cloud services and applications: (1) context-aware access control and policy governance, and, (2) feasible searchable encryption, banded with a distributed key management mechanism.

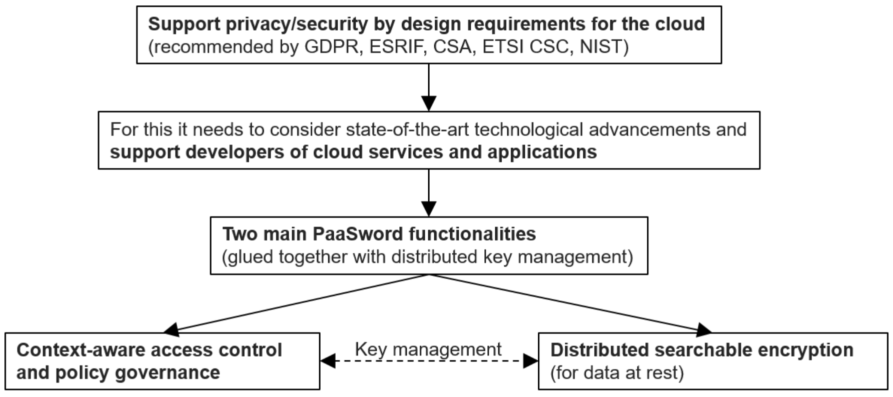
Support privacy/security by design requirements for the cloud
(recommended by GDPR, ESRIF, CSA, ETSI CSC, NIST)
For this it needs to consider state-of-the-art technological advancements and
support developers of cloud services and applications
Two main PaaSword functionalities
(glued together with distributed key management)
Context-aware access control and policy governance
Key management
Distributed searchable encryption
(for data at rest)

Fig. 1. PaaSword baseline motivation and functionality

It relies on recent advancements in the areas of distributed and virtual database encryption and access control modeling and annotation, as well as, on a number of vulnerabilities and challenges posed by international working groups (e.g., NIST, CSA, ESRIF) and European regulations (i.e., EU General Data Protection Regulation (GDPR[8])). The related requirements in GDPR are among the lines of: protection of customer data using appropriate security (e.g., encrypting data to protect privacy), governance tools to safeguard data processing and migration as well as transparent policies to provide

[7] https://www.paasword.eu/
[8] http://www.eugdpr.org/

controlled access to data. The PaaSword framework can assist towards the efforts to complying with certain GDPR stipulations, and in particular with the privacy-by-design stipulation, providing that a suitable set of policies is enforced to meet the regulation requirements.

In this chapter, the project's scientific contribution is provided (Sect. 2) together with the technical solution's high-level description (Sect. 3). In Sect. 4 selected technical components are described in more detail. The selected components explain how context-aware access control rules and policies can be created, managed and enforced, as well as how distributed searchable data encryption and key management are offered by the PaaSword framework. Two cases of preliminary industrial validation (see Sect. 5), results and future impacts in practice are discussed. The chapter concludes with a research agenda (Sect. 6) on future challenges in structuring secure-by-design cloud applications for research and industrial purposes and with final remarks (Sect. 7).

2 PaaSword's Scientific Contribution

Considering both the potential of cloud computing, and the risks associated to it, PaaSword envisages to maximize and fortify the trust of individual, professional and corporate customers of cloud-enabled services and applications. It capitalizes on recent innovations in the areas of: (a) distributed and virtual database encryption and middleware technologies, and (b) access control management and policy governance (including modelling and annotation). The project has contributed to each of these areas with scientific publications. An overview of these contributions is provided in this section.

Regarding the area of distributed and virtual database encryption and middleware technologies, the contribution lies on a scheme consisting of searchable encryption implemented in a database proxy (called DB proxy or adapter), details of which can be found in [7]. The implementation has resulted in a promising concept for preserving security in a reliable manner allowing searching encrypted databases by using indexed keywords. The approach relies on CryptDB [8], one of the most commonly used database encryption schemes and MimoSecco [9]. To the best of our knowledge, none of the public cloud providers offer similar functionality, whereas PaaSword combines a form of searchable encryption together with vertical fragmentation in an innovative manner. Specifically, the searchable encryption approach is an improvement from MimoSecco [9] and the vertical fragmentation is done via a pre-defined set of privacy constraints in order to provide protection against inference attacks on searchable encryption schemes. The DB proxy for secure data outsourcing, trading-off performance and security, is presented in [10], and in [11] vertical fragmentation is added as a privacy-related countermeasure to create the current PaaSword scheme. Moreover, a simple and efficient searchable symmetric encryption (SSE) scheme capable of executing Boolean queries (i.e., multi-keyword queries) on an encrypted Bitmap index is proposed in [12]. This scheme is considered suited for constrained environments where computational power is limited. This means that if the encrypted index is outsourced to the cloud, the scheme may be suitable to run in environments where the communication bandwidth is limited. The reason for this is that during a query execution more than one fragments are utilized to infer the correct result. Employing geographically distributed fragments

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

introduces communication overhead in terms of network delay. As shown for large and dense datasets, such as the Census dataset [13], the encrypted Bitmap index may occupy less storage compared to other known schemes (such as the TSet index used in the BXT and OXT SSE protocols [14]). In addition, the implementation of PaaSword includes advances in the area of key management. Dowsley et al. [15] and [16] present a distributed cloud key management approach. The approach includes encryption of the data by a separate trusted adapter which does not persist the encryption key. The key is split in several fragments and stored in a distributed encrypted way. Details on how bootstrapping, key utilization and key recovery mechanisms work together with security analysis are provided in [16]. State-of-the-art related work is found in [17].

Regarding the area of access control management and policy governance, Veloudis et al. [18] present an automatic mechanism for the validation of policies created to manage the deployment and delivery of dynamic cloud services in highly distributed heterogeneous cloud environments. Data distribution and modelling addressing encryption requirements for secure PaaS-enabled clouds is presented in [19]. In [20], ontological templates suitable for modelling data encryption, as well as data fragmentation and distribution policies, whilst taking into account the underlying domain of application, are presented. In a similar direction, the work in [21] presents an ontological framework for modelling attribute-based access control (ABAC) [22] policies in dynamic and heterogeneous cloud environments. Veloudis et al. [23] present a context-aware security model for cloud applications and outline how policy-related knowledge can be described using an extensible and declarative formalism. The survey on context security policies in the cloud [24] provides an assessment of existing approaches, as well as different modelling formalisms. In [25] and [26] thorough details on the PaaSword framework are presented. More specifically, in these works the PaaSword framework is described focusing on the context-aware security model, the necessary policies enforcement mechanism along with a physical distribution, encryption and query middleware. These works also provide details for guiding developers in defining context-aware rules and policies for protecting sensitive data.

3 PaaSword High-level Architecture

Fig. 2 provides an overview of the PaaSword architecture and depicts the main stakeholder roles. Each of these roles is summarized next.

- **Application User**. This role is dedicated to individuals who use an application deployed on a PaaSword-compliant infrastructure[9]. Most important concerns of this role include: privacy and integrity of the application user data, covering the required functionality of the application and quality aspects, such as performance and availability.
- **PaaSword Administrator/Product Manager**. This role is responsible for maintaining the PaaSword-enabled ecosystem (i.e., the PaaSword context model (see

[9] PaaSword-compliant means that the application is deployed in a PaaSword container which is the operational environment.

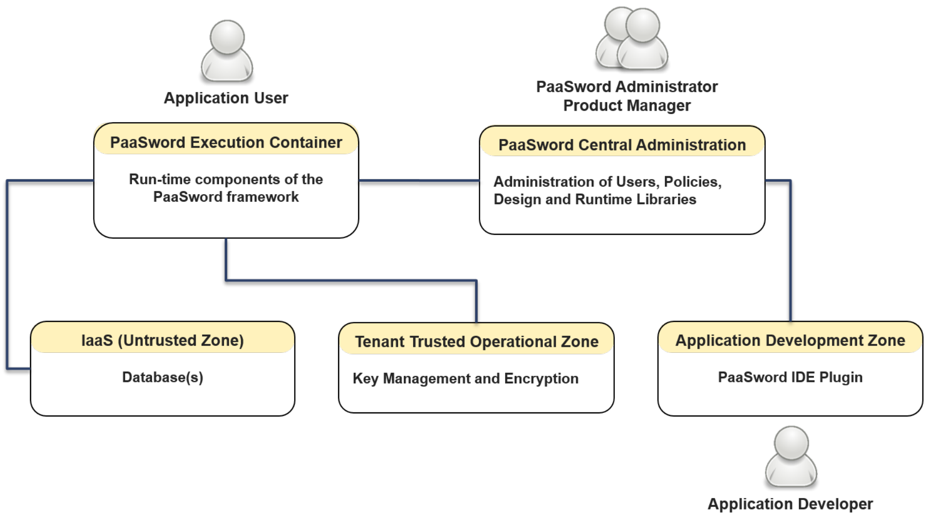
Application User
PaaSword Administrator
Product Manager
PaaSword Execution Container
Run-time components of the PaaSword framework
PaaSword Central Administration
Administration of Users, Policies, Design and Runtime Libraries
IaaS (Untrusted Zone)
Database(s)
Tenant Trusted Operational Zone
Key Management and Encryption
Application Development Zone
PaaSword IDE Plugin
Application Developer

Fig. 2. PaaSword high-level architecture

Sect. 4.1), the PaaSword annotation libraries (see Sect. 4.2) and the various certified enablers used by the PaaS provider). Also, the role is responsible for defining the cloud application architecture and implementation that will meet the requirements of the cloud application. The role also specifies the access policies and the policies for data encryption, fragmentation and distribution.

– **Application Developer**. This role is dedicated to the individual who develops an application that uses the PaaSword components to secure the application. Applications can run on existing PaaS and/or IaaS cloud infrastructures to enhance the data security of the cloud-based applications.

The PaaSword high-level architecture comprises of five main components: (1) an IDE plug-in, (2) a central administration component which handles the PaaSword users and context-aware policies (access control policies, data encryption policies and data fragmentation and distribution policies) and run-time libraries, (3) an execution container for the run-time components of the PaaSword framework, (4) a key and encryption manager, and, (5) the database(s).

The framework covers both a context-aware access policy model and a secure distributed searchable encryption mechanism implementation. The context-aware access policy model provides to Application Developers capabilities for annotating and managing data access controls. The policies are automatically validated, potential conflicts are resolved, and ultimately, they are enforced dynamically. The searchable encryption mechanism improves security by offering fragmentation and reducing the effect of inference attacks. More specifically, the PaaSword framework provides the following functionalities:

(a) An Application Developer may use the PaaSword IDE plug-in to annotate any set of rules required to develop context-aware access policies.

(b) The annotations are dynamically interpreted into policy enforcements.
(c) The PaaSword Administrator/Product Manager and Application Developer may use the PaaSword central administration component to manage rules and policies, and carry out validation checks for the rules and policies.
(d) Access control is bundled with key management and encryption capabilities for securing stored data.
(e) Data distribution is enforced with searchable encryption capabilities to reduce the effect of inference attacks.

The PaaSword framework comprises of the following five layers (shown in Fig. 3):

(1) The first layer, is the PaaSword repository (persistency layer) that represents the technical infrastructure (e.g., repositories), on top of which the PaaSword framework is built. It contains the resources that PaaSword uses for storing (and distributing) encrypted data. It also contains the PaaSword models that serve as the conceptual and modeling pillars used to perform security management of persisted data and policy enforcement of end-users.
(2) The next layer is the core layer that implements the core functionalities offered by the PaaSword framework, such as policy enforcement, policy governance and validity control, annotations interpretation, deployment management, key management, DB proxy for database encryption and distribution.
(3) The PaaSword RESTful API (API layer) that realizes the communication between the back-end services of the PaaSword framework and the front-end UIs.
(4) The PaaSword front-end user interface (UI layer) that supports the interaction between the users and the PaaSword framework functionalities. It includes the context model editor, policy editor, application management, resource management, etc.
(5) The PaaSword libraries and the PaaSword IDE plug-in (third-party layer) that are used in order to enhance an application with the PaaSword framework.

In Sect. 4 the main software components covering some important functionality from the five layers are described.

4 PaaSword Software Components

The PaaSword components, are detailed in this section from the perspective of the three main architectural layers: (i) the UI layer, (ii) the third-party layer, and, (iii) the core layer.

4.1 UI Layer

Context model editor. This component enables all the PaaSword roles (except from the Application User) to use a context model to define all the contextual attributes (e.g., location, device type, time of interaction) whose values need to be taken into account when permitting, or denying, access to a request, or when encrypting a sensitive data object that is to be fragmented and distributed over distinct physical servers for privacy reasons (see Sect. 4.3). The context model editor can also be used to customize

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

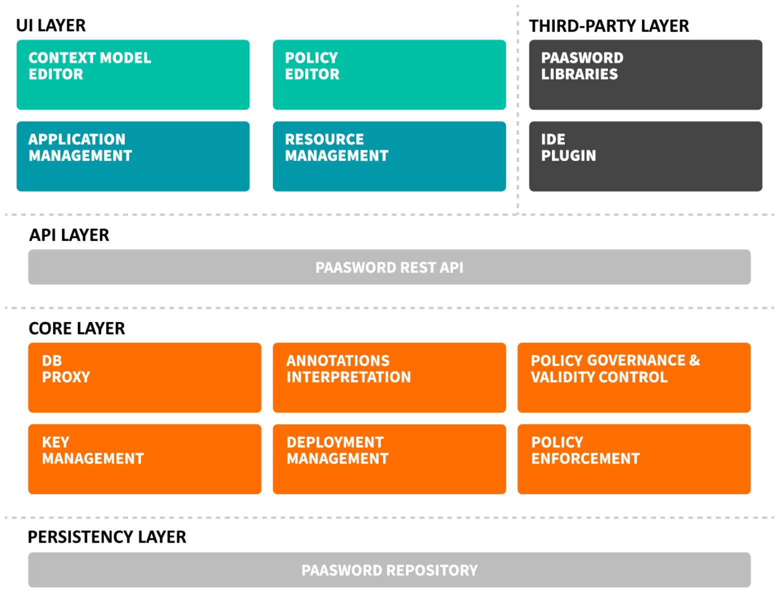

Fig. 3. PaaSword framework layers

the context model according to an organization's business and security needs (e.g., by introducing new contextual attributes or by retiring existing ones). Fig. 4 depicts the abstract classes of the context model. Implementation details can be found in [23] and [27].

Policy editor. This component enables all the PaaSword roles (except from the Application User) to create, delete or update rules, policies and policy sets. The definition of policies is based on the eXtensible Access Control Markup Language (XACML) [28], a declarative fine-grained, attribute-based access control (ABAC) policy language. The PaaSword implementation carries out at design-time automatic validation of the formulated policies. XACML supports three granularities of authorization entities. These include: (a) the rule, (b) the policy which is a conceptual aggregation of rules, and (c) the policy-set which is a conceptual aggregation of policies. Each of these entities are supported through dedicated editors developed in the PaaSword framework. In addition, an expression editor is developed to allow for the creation of context expressions that draw their attributes from the underlying context model, as well as for the definition of custom combining algorithms for determining to which one of its constituent rules a policy resolves. Implementation details can be found in [26], [27] and [29].

4.2 Third-party Layer

PaaSword libraries. The PaaSword library (referred to also as libraries) is used by the Application Developer for the integration of PaaSword within an application. The

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

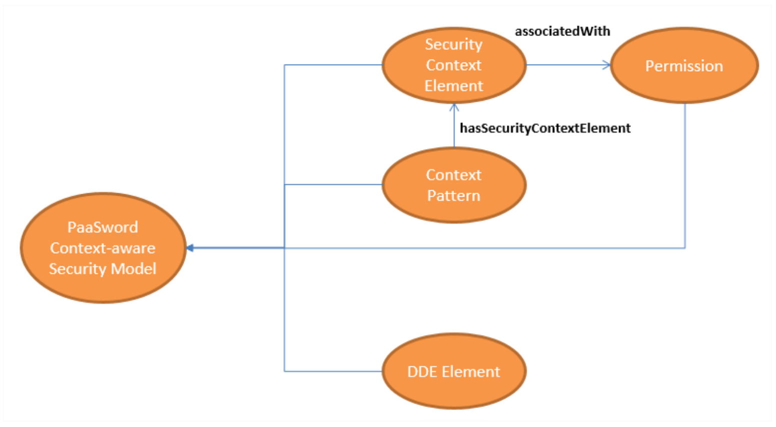

Fig. 4. PaaSword context-aware meta-model

library offers both the appropriate annotations and executable code, such as the authorization request, that handles the communication between the application to be secured and the relevant components of the PaaSword core layer (as described in Fig. 3).

Annotations: Three annotation types are provided by the PaaSword library which the Application Developer can use to secure an application: PaaSwordPEP, PaaSword-DDE and PaaSwordEntity. They are used to mark relevant entities that are later parsed by an introspection mechanism (byte code analysis) during run-time and made available for modeling within the PaaSword UI (where context-aware access control, data distribution and data encryption rules can be defined).

PaaSwordPEP is used on a class or method level to define Policy Enforcement Points (PEP). A PEP represents either a policy or a policy set containing rules and an entry point of validation and enforcement. Within a Spring project[10], this annotation can also be used together with an interception mechanism based on Spring AOP[11].

PaaSwordDDE is a class-level annotation for Data Distribution and Encryption. The encryption algorithm to be used can be specified as a parameter value of the PaaSwordDDE. During the introspection this value is loaded and added to the application's data encryption and fragmentation configuration.

PaaSwordEntity is a class-level annotation used to define what data object types will be handled by the DB proxy data fragmentation and encryption mechanism (see Sect. 4.3).

Authorization request: The PaaSword library includes a service which provides the authorization request. It allows any caller to initiate an authorization request handled by the PaaSword controller and delegated to the Semantic Authorization Engine (see

[10] Implemented using the Spring Framework (https://spring.io)

[11] Implementation of the Aspect-Oriented Programming paradigm by Spring (https://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html)

Sect. 4.3). The possible conclusions from the engine, i.e., 'permit'and 'deny'are then propagated back to the caller for carrying out the application-specific evaluation. The authorization request is also used by the interception mechanism that the PaaSword library provides. In the latter case, the necessary method argument values are resolved by reflection from the PaaSwordPEP annotation and the scope of the method call. When used without the introspection mechanism of PaaSword, the method call usually happens within PaaSwordPEP annotated methods. A more detailed account of the authorization request can be found in [29].

4.3 Core Layer

Policy governance and validity control. As its name suggests, this component provides a policy governance mechanism, one that is underpinned by an ontological representation of policies that semantically describes the various knowledge artefacts that are incorporated in the policies. In this respect, it promotes a clear separation of concerns by disentangling the definition of policies from the code of the applications into which they are infused. This enables the provision of the following seminal capabilities in a generic and automated manner: (i) Reasoning about the well-formedness of the policies, by constraining the knowledge artefacts that must, may, or may not be embodied in the policies. (ii) Determining potential inter-policy relations, such as subsumption and contradiction, that may affect the effectiveness of the policies. (iii) Enabling the performance of policy life-cycle actions, such as policy updates and retirements, in a controlled and rule-based manner, as well as determining the repercussions of such actions on the overall ability of the policies to protect critical assets. More details can be found in [18], [20] and [21].

Annotations interpretation. This component is responsible for the interpretation of annotations into XACML-based enforceable access control policies. It implements the following functionalities: (a) Introspects the source code of a PaaSword-enabled application to determine whether it contains appropriate PaaSword annotations or not, (b) Feeds the PaaSword policy enforcement mechanism with the appropriate rules, (c) Informs which attribute values are needed by the policy enforcement mechanism for resolving a specific access request, and, (d) Feeds the mechanism with the appropriate current values of the needed attributes that will allow the successful evaluation of policy sets. For the types of annotations and their usage see Sect. 4.2, also more details are in [27] and [29].

Policy enforcement. This component determines whether an authorization request to access a particular data object should be permitted on denied. This determination is based on a set of access control policies that are persisted in the PaaSword repository (see Fig. 3), as well as on the values currently assumed by all those attributes that need to be taken into account in order to decide whether to permit or deny a request. A subcomponent of the policy enforcement component is the Semantic Authorization Engine.

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

Semantic Authorization Engine: The engine carries out pattern matching of real-world contextual facts and data with rules to infer whether to permit, or deny access requests. These rules are derived from the access control policy rules. During an access request, the contextual information that is required for the rules evaluation is collected by specific mechanisms (called “context handlers”). The contextual information is transformed to semantic facts which are inserted in the engine’s working memory. During the insertion of the facts, additional facts can be introduced using semantic reasoning rules (such as sub-classing, generalization etc.). The introduced semantic facts trigger the authorization rules which infer whether the request can be granted or not. In cases where a number of rules and facts need to be processed, and a number of these rules might be true for the same fact assertion, then these rules are said to be in conflict. Then, the execution order of these rules will need to be defined based on a conflict resolution strategy. More details may be found in [26], [27] and [29].

Database proxy. The DB proxy design proposed in PaaSword enhances the proxy introduced in the research project MimoSecco[12][9]. Details on the DB proxy design can be found in [29]. Consider the case of database outsourcing with a trusted entity (the DB proxy) that is located between the user and an untrusted Cloud Service Provider (CSP) (see Fig. 5). The chosen approach for the DB proxy aims to transfer as little data as possible. Therefore, we use additional index data structures to determine which data segments seem relevant to the query, relying on the relational model of structured data. The implementation is able to handle complex realistic SQL queries.

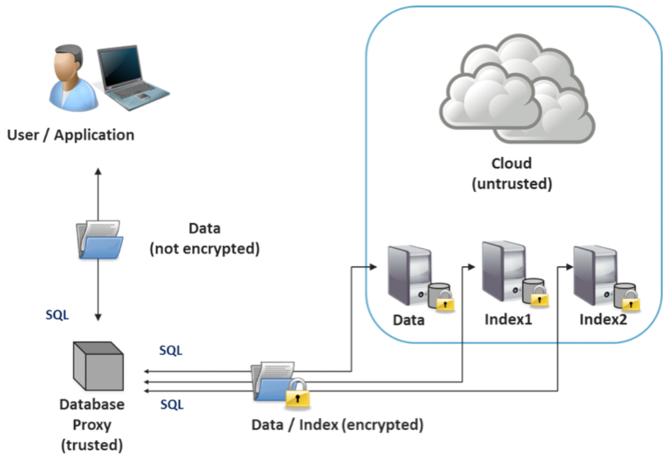

Fig. 5. The PaaSword DB proxy

The DB proxy supports the implementation of transparent distribution and encryption/decryption. Consider the example database table (Table 1).

[12] https://sites.google.com/site/mimoseccoproject/

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

Row	Name	Surname	Condition
1	Claire	Jones	Diabetes
2	Emilia	Jones	Hypertonia
3	Phil	Connor	Hypertonia

Table 1. Original data before transformation

Row	Encrypted Data
1	Enc(Claire‖Jones‖Diabetes)
2	Enc(Emilia‖Jones‖Hypertonia)
3	Enc(Phil‖Connor‖Hypertonia)

Table 2. Encrypted data table after transformation

Attributelist	Rowlist	Attributelist	Rowlist
0x0B34 = DetEnc(Phil)	Enc(3)	0x5AFE = DetEnc(Jones)	Enc(1‖2)
0x78AB = DetEnc(Claire)	Enc(1)	0x7732 = DetEnc(Connor)	Enc(3)
0xC134 = DetEnc(Emilia)	Enc(2)		

Table 3. Index tables after applying the PaaSword transformation with encrypted index

The mechanism operates both at the row and the column levels of the database and produces (at least) two results (fragements). The first one (see Table 2) uses randomized encryption on the stored data. All the attribute values (data from the rows) are concatenated and put into a single ciphertext. The second result is a table called "index table" (Table 3). It consists of a quick look-up of links to the position of certain attribute values, i.e., where the attributes occur in the original table. The positions are probabilistically encrypted. The attribute values can be deterministically encrypted (as in Table 3) or remain in plaintext. Two ways of distributing fragments are supported:

1. Distribution may be carried out on each index for every column to a single server. Even if this solution enhances security, it requires a lot of servers, thus it is not scalable.
2. Use multiple index servers, but distribute multiple indexes to a single server according to pre-defined privacy constraints that break the association between the previously related database columns. So besides enhancing security, the privacy constraints allow us to accommodate more than one column in each server.

Deterministic index encryption: The DB proxy achieves IND-ICP (Indistinguishability under Independent Column Permutations) security [30], it builds on top of the well-established IND-CPA (Indistinguishability under Chosen-Plaintext Attacks) security [31] and guarantees that the relation among the attribute values is not leaked. By encrypting the attribute values in the indexes, it is more difficult for the adversary to perform side-channel attacks. Using deterministic encryption, many queries such as exact-match queries can still be performed as the DB proxy can re-create the corresponding ciphertext used in the index. Thus, deterministic index encryption is an appropriate approach that achieves a good trade-off between security and efficient support for as many different query types as possible.

For example, when the database adapter receives a query for the name "Phil", it encrypts "Phil" with the deterministic encryption scheme in order to obtain "0x0B34"

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

and then searches for this value in the index table. As a result "$\mathsf{Enc}(3)$" is obtained, i.e., the encrypted link (using a randomized encryption scheme) to the row corresponding to Phil. The adapter then decrypts the value to 3 and requests the third row from the encrypted data table. Upon receiving the encryption $\mathsf{Enc}(\text{Phil}||\text{Connor}||\text{Hypertonia})$ (where $||$ denotes concatenation), the adapter can decrypt it to get Phil's attributes. Negation (of statements) and "IN" queries (which allows to match multiple values) can be supported in a similar way using standard set operations.

Distribution via privacy constraints: One can improve the security strength of the above approach for searchable encrypted relational databases (i.e., protecting a relational database using searchable encryption) by distributing the columns of the relational database under consideration into several fragments using privacy constraints. Note that the privacy constraints, as defined in [32–34], were mainly used to evade the use of encryption or to use encryption as less as possible. However, in PaaSword, we propose using them to strengthen the security of the SSE schemes against inference attacks. More details about the security gain when distributing a searchable encrypted relational database using vertical fragmentation via pre-defined privacy constraints, can be found in [11].

Frequency-hiding encrypted inverted index: The index encryption approach of the PaaSword DB proxy reveals the frequency (i.e., result length) of encrypted attribute values before being queried but it does not reveal the joint frequency between encrypted attribute values. One can easily hide the frequency of an encrypted attribute value before being queried by modifying the above encrypted index. There are several approaches to do this frequency hiding modification. Chase and Kamara propose a SSE scheme that pads all posting lists up to the frequency of the most frequent keyword before encrypting them [35]. Another approach proposed in [36] is to have more than one row per keyword. This will of course increase the latency but the size of the encrypted index will be optimal. One can see that hiding the frequency of encrypted keywords in the encrypted index comes at the cost of efficiency. Therefore, for efficiency reasons, PaaSword chooses to hide only the joint frequency between encrypted keywords by encrypting the corresponding document/record IDs of each encrypted keyword. According to Cash et al. [37], the encrypted inverted index with no frequency hiding can be considered under the same leakage profile of fully frequency-hiding searchable symmetric encryption schemes which are provably secure under Curtmola et al.'s security model [38].

Key management. The key management component is responsible for defining the mechanism for preparing, distributing and using the keys for database access, during application bootstrapping and run-time. Consider the case of an Application (A) hosted by a third-party (i.e., an application service provider (ASP)) which manages data of a tenant (data owner). For security reasons, data should be stored encrypted. In addition, access should only be made possible with the consent of the tenant. Therefore, the key to access the stored encypted data is called "tenant key" (TK) and should be only under the control of the tenant.

A common approach is to store the TK where the database is accessed (at the ASP). In this way, however, operators of the ASP gain access to the data at all times without any control by the tenant. To avoid this security problem, one known approach is to isolate A logically and physically from administrative access by operators during run-time. Before granting this administrative access, all tenant data need to be encrypted and stored in a secure location (i.e., a location where only the tenant has access). Another solution to give the tenant more controlled data access, is to have a specific key server operated by the tenant. Every access to the data needs to fetch the key from the tenants key server, which requires an authorization before revealing the key. To avoid the necessity of operating a key server, another possibility is to secure the key by storing it inside a secure hardware revealing the key only upon tenant authorization.

The DB proxy implementation of PaaSword (see Sect. 4.3) offers a solution for securing data at rest stored by an untrusted third-party, like a CSP. In many cases, it is necessary that different users require access to the data. If the data are encrypted, every user needs access to the database encryption and decryption keys. An insecure solution is to give the keys to every user. A better solution is to store this key inside the DB proxy which performs the encryption and decryption transparently. However, this means that the administrator of the DB proxy has uncontrolled access to the data. Moreover, if the database key is stolen by an adversary, the adversary gains unlimited access to the data.

To reduce this risk, we introduce a distributed key management scheme which splits the database key in different distributed fragments and composes them only when and where needed. The PaaSword approach avoids operating a key server or the use of specific secure hardware. It separates the Application (A) where the data is processed from the DB proxy (P), whose task is to store and access the data in a cloud database on behalf of A when authorized by a User (U_i). Fig. 6 shows the key management mechanism during run-time. U_i encrypts his individual part of the key (TKu_i) together with a timestamp using the public key of P (referred to as Enc_P $(TKu_i \| time)$) and adds it to his request to the application for processing data. A controls the permission of the user to process the data and, if permission is granted, A can use Enc_P $(TKu_i \| time)$ and its own user specific part of the application key (TKa_i) to request the necessary database operations from P. P decrypts Enc_P $(TKu_i \| time)$ and controls the timestamp. If the timestamp is within a defined validity period (e.g., 5 minutes), it reconstructs TK = $TKu_i \oplus TKa_i \oplus TKp_i$, checks the validity of TK and performs the requested database operation. Afterwards P wipes TK out of its memory.

This splitting of the key also ensures that the access control mechanism cannot be bypassed in any way, because the access control mechanism is in possession of one of these key parts. The CSP is used only as a storage back-end and is not required to participate in the key management. Our key management scheme protects the databases confidentiality even if the key shares of an arbitrary number of users and the application are compromised and does not rely on heavy tools like a global PKI (Public Key Infrastructure) for user keys. This offers the advantage of a very lightweight deployment.

However, it is still necessary to have one location where the complete database key is stored. For this we use a component called “Tenant-Admin” which is needed to set up the process. Afterwards the database key should be removed from this component

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

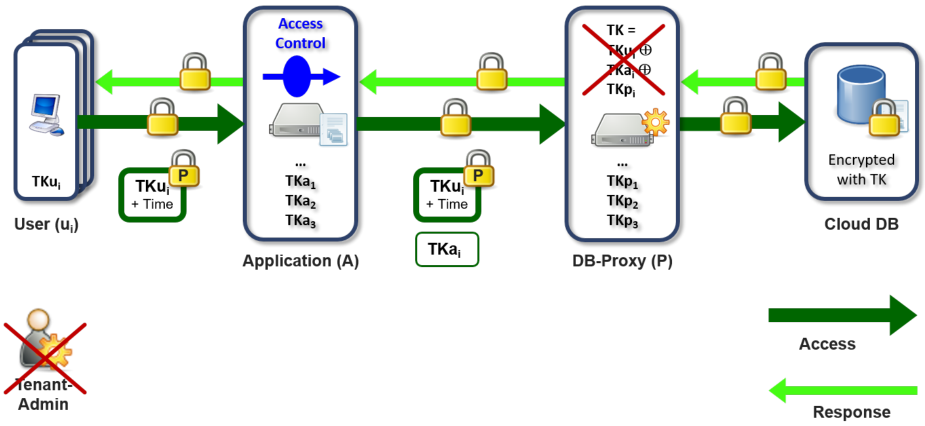

Fig. 6. Key management mechanism during run-time [15]

and stored in a (physically) safe place. The Tenant-Admin with the database key is only needed again to add users or for key recovery reasons.

The PaaSword approach suggests to split the database key into three parts. However, it is possible to extend the mechanism to more than three parts if more instances exist. It can also be reduced to two parts, leading to lower security, because it is only necessary to compromise two instances. A detailed explanation of PaaSword's key management approach can be found in [15]. More details of how certain mechanisms, especially bootstrapping, key usage and recovery work, together with a security analysis are described in [16].

5 Preliminary industrial validation with pilots

5.1 Pilot: personal data protection in a multi-tenant CRM environment

A pilot integrating the PaaSword components is implemented in the industrial setting of a German software company. The pilot carries out validation through realizing personal customer data protection in a real multi-tenant Customer Relationship Management (CRM) environment. More specifically, the validation's aim is to examine the practicability, applicability and overall impact of the PaaSword framework on the competitive advantage it offers. The pilot scenario follows through the process of granting or restricting access to real customer data in a secure way based on specific policies defined through the PaaSword framework. Fig. 7 depicts the PaaSword-enabled CRM cloud platform supporting two possible access rights paths: (i) a "regular", and (ii) an "enhanced" one, supported by PaaSword. The overall result of whether an access should be allowed or not, is seen as the logical conjunction of these two paths.

The pilot integration is using SmartWe[13], a Java-based CRM cloud platform developed in-house by the software company, and selected PaaSword components. Specif-

[13] http://www.smartwe.de

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

ically, the PaaSword components used are: (i) the PaaSword core layer components (except the DB proxy and the deployment management component), (ii) the context model and policy editor to allow maintenance of both model and policies from the UI layer, and (iii) the PaaSword libraries, as the base component, to secure an application with PaaSword. Details on the technical implementation with examples are described in [39].

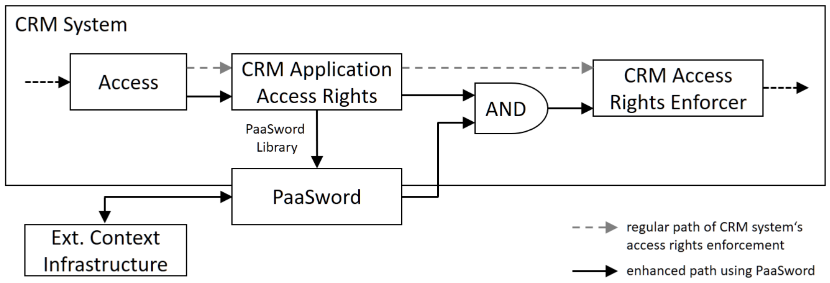
CRM System
Access
CRM Application Access Rights
PaaSword Library
AND
CRM Access Rights Enforcer
PaaSword
Ext. Context Infrastructure
regular path of CRM system's access rights enforcement
enhanced path using PaaSword

Fig. 7. Pilot implementation: The PaaSword-enabled CRM cloud platform

Preliminary results. The practicability of the PaaSword framework through the integration steps carried out is examined first. The integration of the components via the PaaSword library component into a productive environment was seamlessly performed. The main reasons for this are: (i) compatibility with the existing Maven[14] dependency management system, (ii) easy to introduce workarounds offered by the PaaSword framework in order to circumvent the necessity of Spring AOP, (iii) a multi-functional web-based UI exists, (iv) annotations could be easily used by the developers, and, (v) ability of carrying out installations for multiple tenants.

The validation also examines the applicability of the PaaSword framework, i.e., whether the use of the PaaSword components was well-suited to the pilot's needs. The model-based and policy-driven context authorization mechanisms of PaaSword are key to the pilot's integration scenario, since the focus here lies on the restriction of the execution of the business logic (e.g., a document upload) based on a user's context. This could be achieved with low effort by: (i) extending the existing code base of SmartWe with annotations provided by the PaaSword libraries, (ii) adapting the context model using the PaaSword context model editor, and (iii) the creation of adequate policies using the PaaSword policy editor. Both editors allowed for an immediate and straightforward skeleton setup of the validation scenario intended. Since the pilot targeted the protection of sensitive customer data in a multi-tenant CRM platform, the segregation of PaaSword into two executables (core components and Semantic Authorization Engine) allowed for a flexible and tenant-wise deployment.

Finally, the validation aims at assessing the overall impact of the PaaSword framework on the competitive advantage that the adoption of PaaSword offers. The pilot

[14] http://maven.apache.org/

integrating the PaaSword components offers a competitive advantage over other CRM vendors in the business domain, especially regarding the security and data privacy-enabling features it enhances. It is particularly beneficial due to the model-based context authorization engine and the key management mechanism offered. They represent promising innovative technologies that are integrated into the company's products and at the same time translated to internal expert knowledge. Thus, this paves the way for future evolutions of the CRM solution, leading to offerings that allow customers from small and medium enterprises (SMEs) to enhance security and take initial actions towards compliance with data privacy demands (i.e., the compulsory compliance to the EU GDPR).

5.2 Pilot: secure transportation data fusion and analytics

A second pilot implementation making use of the PaaSword framework is providing validation of secure data fusion and analytics functionalities according to a Smart City scenario. Within this scenario, the existence of a large, heterogeneous sensor infrastructure is assumed. This system's task is to supply suitable information to various consumers and services providing monitoring and controlling support. The scenario is supporting the case of Smart Goods Transportation services, which rely on stationary and mobile sensors for actively tracking and reporting the state of transferred goods through the entire logistics chain, from their dispatch and up to their reception. The type of information generated by the sensors and attached to deliveries, as well as the heterogeneity of the available sensors and entities interacting with the goods, have strong security and privacy requirements. These relate for example to the contents of the goods, their value, the interaction type and the frequency of communication between the sender and the receiver, since all represent confidential information. The pilot's architecture consists of three layers (detailed in [39]):

- A sensor and actuator layer, consisting of a plethora of heterogeneous devices addressable over standard Internet of Things (IoT) protocols.
- A cloud-based middleware, which persists, aggregates and processes data in order to detect certain trends and generate predictions in order to enable a proactive reporting, handling and diminishing of occurring risks.
- An application layer, which exposes the middleware functionalities to various end-users (e.g., services or individuals).

The integration of the pilot with the PaaSword framework is carried out through the use of the PaaSword libraries. Two main PaaSword components are employed: (i) the policy enforcement, acting as an enforcing point of the access models defined by the various entities interacting (either as users or as data providers) with the pilot and which relies on the annotation interpretation mechanism, and (ii) the DB proxy, which is used to transparently encrypt the data.

Preliminary results The preliminary validation focuses on three main characteristics of the PaaSword framework: (i) usability of the framework, (ii) configurability, and, (iii) added value in the context of addressing the security and privacy requirements.

The usability refers to the ease of integration of the PaaSword components into existing software platforms. The configurability stems from the need of having to tailor security and privacy policies to possible scenarios with varying requirements to meet different regulations and customer specifications. Thirdly, addressing security and privacy requirements through mechanisms which allow users of the framework to maintain data ownership by employing cryptographic mechanisms providing security guarantees is an approach for mitigating potential exposure risks, but also mainly a way of providing to the users visibility of their data usage.

The annotation-based approach in integrating PaaSword's policy enforcement points (PaaSwordPEP) required minimal efforts for its integration into the existing code bases. More specifically, the pilot did not require any change of the existing code besides from the addition of method-annotations. It required the creation of the specific handlers for extracting the context-relevant information used by the various policies from specific requests (such as the OS information, geographical location, device type, etc.), which however required a low effort to accomplish. The usability characteristic, therefore, quantified as effort spent, is achieved, i.e., little effort was needed. Another important aspect was that the integration did not require extensive testing, due to the legacy code not being changed substantially.

The second main characteristic, namely configurability, is reflected in the context of this pilot as the ability of the PaaSword models and (policy) expressions to be easily adapted to different enforcement scenarios. More specifically, the adaptation of the context model as well as the definition of the expressions employing it in order to define relevant access policies for domain- and application-specific usage within the pilot. These include the definition of new device type classes such as gateways (sometimes revered to as sinks, representing components connecting sensor networks with the IoT cloud middleware), device deployment type (private or public), or classes reflecting the mobility and class of the infrastructure element within which sensors can be embedded (e.g., trucks, warehouses). Since the expert knowledge required by the PaaSword Administrator, Product Manager or Application User for adapting the models and policies for specific scenarios did not require any new software code to be written, changes could be performed and easily and formally verified through the mechanisms provided by the framework itself (i.e., through the Policy governance and validity control). Similarly, the decoupling of the models and policies from the deployed applications and the run-time adjustment to any changes both provide a highly configurable environment.

Lastly, the transparent encryption mechanisms – used in conjunction with the key management mechanism, provides a strong case for self-data governance, a strong requirement for many of the private and industrial users when having to store sensitive data in cloud platforms. Towards this goal, the pilot is experimenting with using a transparently encrypted NoSQL database for storing high volumes of (volatile) data, as well as with a custom key management mechanism, employing anonymity-preserving operations suitable for autonomous IoT sensors as those used in the pilot (i.e., secret handshakes and zero-proof of knowledge instead of the interactive process typically used in human user-initiated interactions).

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

6 Research Agenda

A research agenda has been developed to present our future work direction and it has been divided in the following sections: (i) data privacy, (ii) reliable security and (iii) usability. We highlight challenging research areas for the project continuation and identify areas of future contributions.

6.1 Data privacy

– **Secure execution of range queries:** One approach to perform range queries is to use Order Preserving Encryption (OPE) [40, 41]. However, recent attacks on OPE schemes suggest that they are insecure [42, 43]. Moreover, some recently developed generic attacks target any encrypted database scheme that supports range queries but leaks the access pattern without assuming that the attacker has a prior knowledge about the database under attack [44]. Most notably, one of the generic attacks in [44] target even secure encrypted search methods supporting range queries and only leaking the communication volume (i.e., the query result size), such as fully homomorphic encryption [45] or ORAM schemes [46]. More recently, the generic attacks in [44] have been improved in [47] where the full access pattern leakage is exploited. All these attacks confirm that constructing a secure database scheme supporting range queries functionality is a challenging task. Preventing the attacks exploiting only the result size is challenging but note that theses attacks assume that the user's queries follow a uniform distribution which might not be true in the real world. Therefore, the main threat comes from attacks exploiting the full access pattern leakage. However, one approach to prevent the attacks exploiting the full access pattern leakage is to hide the access pattern leakage using techniques such as ORAM [46] but still ORAM techniques are not scalable despite the recent development in [48]. As such, finding a scalable access pattern hiding technique is an interesting open problem that needs to be addressed by the research community.
– **Forward privacy schemes:** Forward privacy means that an update procedure should not leak if a newly inserted record or document matches previous search queries [49]. Such a property prevents the recent devastating adaptive file or record injection attacks on SSE schemes [11, 50]. Another enabled feature about forward private schemes is that they allow the secure and on-line construction of an encrypted database index (this is to be contrasted with the PaaSword approach whereby the encrypted database index is constructed off-line and then outsourced to the cloud). Current forward private SSE schemes have efficiency problems which makes them not scalable for processing large databases. For example, the SSE schemes in [51, 52] based on ORAM and in [49] based on asymmetric cryptography, do not offer a suitable locality. Thus, designing a scalable forward private scheme is an interesting and challenging open research problem that needs to be addressed to increase both the privacy and efficiency of searchable encryption schemes.
– **Execution on trusted hardware:** As a central component of PaaSword, the DB proxy handles the user queries to the encrypted database and has full access to the database. As a consequence it also has access to the encryption keys. In the PaaSword context, cloud providers are considered to be honest-but-curious (i.e.,

would not affect data integrity). This precludes the deployment of the DB proxy in the cloud. Thus, the use of standard untrusted hardware and software would not prevent a malicious provider from monitoring the DB proxy's execution and would then gain access to the raw database contents as well as the encryption keys. This, however, can be solved with the use of hardware which provides a trusted execution environment (TEE). TEE features isolation which prevents an adversary who is in full control of the system from reading the TEE's memory containing plaintexts and encryption keys [53]. Also, TEE offers remote attestation, i.e., it allows a third-party to first verify the integrity of the deployed code and that it runs in a TEE, and then deploy sensitive data (such as encryption keys) through a secure channel established during attestation [53]. For example, with the introduction of Intel's Software Guard Extensions (SGX)[15] in the Skylake processor family, such TEEs come in mainstream hardware. In contrast to existing solutions, like the ones based on TPMs (trusted platform modules), they offer the advantage of considering a much smaller trusted computing base. This base consists of namely the application and its libraries (in contrast to the whole system) and provides stronger security guarantees, like encrypted memory or isolation from components for example the BIOS or the management engine [54].

6.2 Reliable security

- **SQL expressiveness:** When offering database access through an adapter, SQL expressiveness is always in doubt. The DB proxy in PaaSword performs a lot of query rewriting and data transformations. Even in the absence of implementation bugs, there is still lack of support for some query types. Consider as an example a fully encrypted index that cannot provide an efficient way to run wildcard search queries (SQL's *LIKE* statement with wildcard search) on encrypted data. Consequently, all future queries have to be taken into account during design-time when an encrypted index is considered. In PaaSword, we have analyzed through real use cases the most frequently used SQL queries from an industrial perspective, taking into consideration the intended support for the pilots and intentionally chose to support a specific set of SQL statements. The implementation is compliant to the most important non-trivial challenges, for instance recursive sub-selects and joins of all kinds, but nevertheless ample room for additions exists. For instance, some language features like the *autoincrement* are not handled. To give a full support for the SQL vocabulary future work could address all remaining unsupported statements and enable for richer queries.
- **Data integrity:** The PaaSword approach currently considers only passive attackers (i.e., attackers who are honest-but-curious). However, in order to protect PaaSword against a malicious server who can tamper with the search results, we need to allow the client to verify the search result. This will enable the client to notice any malicious behavior from the server. Such a functionality can be achieved through the use of verifiable searchable encryption schemes [55, 56] which employ Message Authentication Code (MAC) algorithms to provide integrity to the search results.

[15] https://software.intel.com/en-us/sgx

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

– **Database compatibility**: Back-end independence is importnant when introducing an additional middleware layer. In the case of PaaSword, the DB proxy is such a middleware that resides between the application and the storage back-end. To have minimal impact on the application, the middleware has to act like a normal relational database. Different DB back-ends, however, are not fully compatible, even if SQL is used in a conservative way. Various relational database management system (e.g., MSSQL, MySQL, Oracle and PostgreSQL) act differently. Hence, the DB proxy needs to emulate the specific database management system behavior. In PaaSword, we chose Postgres as the DB back-end. The support of other relational database systems, especially the widely-used MySQL, is a future task that could provide to the PaaSword framework better compatibility and an even higher impact.

6.3 Usability

– **Automation of the distribution process:** In the case of a relational database with a large number of columns (e.g., 40 columns or more) it can be challenging for the PaaSword Administrator/Product Manager to define the privacy constraints needed in order to perform vertical fragmentation. An interesting future research direction could be to investigate whether machine learning or Natural Language Processing (NLP) or deep learning techniques can enable us to automatically define the privacy constraints of a given relational database table. For instance, in [57, 58] NLP has been used recently to extract access control policies from natural language documents. Such directions may allow us to automate the process of vertical fragmentation and thus make it easier to use it, regardless of the number of columns of the relational database table under consideration. As systems are growing in terms of storage and processing capacity such features are considered mandatory to be supported in the future.
– **Key management handling:** The concept of PaaSword demands that the user parts of the database key are distributed from the tenant admin to the user devices via an USB flash drive or another offline method. This concept is easy to set up and has no additional requirements for the clients, but it lacks usability. To avoid using complex tools, like a PKI, a comfortable lightweight solution for the distribution of the user parts of the database key is needed. Such extension to the PaaSoword work could be pursued in the future work.

7 Conclusions

Concurrently with the various efforts to offer advances on the topic of security and privacy of data carried out at international and pan-European level, the PaaSword project offers a data privacy and context-aware security-by-design framework for software developers. It addresses cloud security threats introduced by i.e., data access breaches, insecure APIs and weak or inadequate mechanisms for access control and encryption. The PaaSword project offering is accompanied by several technical and scientific contributions presented in this chapter.

Specifically, one of the main offerings of PaaSword is to ensure the creation of context-aware rules and policies as well as utilization of automatic mechanisms for the

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

validation of the policies to optimally manage, deploy and deliver dynamic cloud services in highly distributed heterogeneous environments. The approach enables security annotations to be created transparently and effectively applied through usable libraries. The annotations are then transformed into context-aware rules and policies which enforce access control, cryptographic protection, searchable encryption and fragmentation for securing sensitive data at rest. The PaaSword framework ensures protection and privacy for cloud user data and enables security-by-design to software applications.

Details of the technical solution proposed within the project are presented through the descriptions of the PaaSword components belonging to three main architectural layers. The concepts included in the implementation of the PaaSword framework are presented, as well as the benefits achieved. Moreover, the concepts introduced in the PaaSword framework have been validated through two industrial pilot implementations. They carry out an initial validation step for the benefits and opportunities offered by the PaaSword framework. The industrial pilots are focused on examining the usability, applicability, configuration ability and overall competitive advantage the framework offers. To the best of our knowledge, the combination of the above mentioned concepts in real applications have not been employed in other contexts. The projects' implementations aim to validate that cloud security can be increased without major concerns on performance, reliability and availability.

Finally, further efforts to increase security in the cloud are required to address issues identified as future work or challenging areas of research interest in the research agenda.

Acknowledgements

The authors wish to thank for their input all the researchers and experts working in the *PaaSword* project. We especially thank Dr. Simeon Veloudis for comments that greatly improved the manuscript. The work is supported by the European Unions Horizon 2020 research and innovation programme under grant agreement No *644814*.

References

1. Gartner: Gartner says worldwide public cloud services market to grow 18 percent in 2017. http://www.gartner.com/newsroom/id/3616417 (2017) Accessed: 2017-09-10.
2. ESRIF: European Security Research and Innovation Forum final report. https://ec.europa.eu/home-affairs/sites/homeaffairs/files/e-library/documents/policies/security/pdf/esrif_final_report_en.pdf (2009) Accessed: 2017-09-10.
3. Ross, R.S., McEvilley, M., Oren, J.C.: Systems security engineering: Considerations for a multidisciplinary approach in the engineering of trustworthy secure systems. Special Publication (NIST SP)-800-160 (2016)
4. ISO/IEC/IEEE: 15288:2015 Systems and software engineering – System life cycle processes. (2015)
5. CSA: (Cloud Security Alliance) top threats Working Group: The treacherous 12 cloud computing top threats in 2016. (2016)
6. Walker, K.: Cloud Security Alliance releases the treacherous twelve cloud computing top threats in 2016. https://cloudsecurityalliance.org/media/news/cloud-security-alliance-releases-the-treacherous-twelve-cloud-computing-top-threats-in-2016/ (2016)

Original publication:
European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017, 3-27, 2017, Porto, Portugal, DOI:10.5220/0007901500030027

7. Abdelraheem, M., Dowsley, R., Gabel, M.: Distribution algorithms and encryption schemes - Deliverable D4.1. https://www.paasword.eu/wp-content/uploads/2016/11/D4-1_DistributionAlgorithmsAndEncryptionSchemes_forSubmission_final.pdf (2016) Accessed: 2017-09-10.
8. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ACM (2011) 85–100
9. Achenbach, D., Gabel, M., Huber, M.: MimoSecco: A middleware for secure cloud storage. In Frey, D.D., Fukuda, S., Rock, G., eds.: Improving Complex Systems Today, Proceedings of the 18th ISPE International Conference on Concurrent Engineering, July 4-8, 2011, Boston, MA, USA, Springer (2011) 175–181
10. Dowsley, R., Gabel, M., Yurchenko, K., Zipf, V.: A database adapter for secure outsourcing. In: Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on, IEEE (2016) 503–508
11. Abdelraheem, M.A., Andersson, T., Gehrmann, C.: Searchable encrypted relational databases: Risks and countermeasures. In: Proceedings of the Twelveth International Workshop on Data Privacy Management (DPM 2017), Springer (2017)
12. Abdelraheem, M.A., Gehrmann, C., Lindström, M., Nordahl, C.: Executing boolean queries on an encrypted bitmap index. In: Proceedings of the 2016 ACM on Cloud Computing Security Workshop, ACM (2016) 11–22
13. Lane, T., Kohavi, R.: Census-income (kdd) data set. https://archive.ics.uci.edu/ml/machine-learning-databases/census-income-mld/ (2000) [Last Accessed June 2017].
14. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Advances in cryptology–CRYPTO 2013. Springer (2013) 353–373
15. Dowsley, R., Gabel, M., Hübsch, G., Schiefer, G., Schwichtenberg, A.: A distributed key management approach. In: Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on, IEEE (2016) 509–514
16. Schiefer, G., Citak, M., Schoknecht, A., Gabel, M., Mechler, J.: Security in a distributed key management approach. In: IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS), IEEE (2017)
17. Michalas, A.: PaaSword technical requirements - Deliverable D1.1. https://www.paasword.eu (2015) Accessed: 2017-09-10.
18. Veloudis, S., Paraskakis, I., Petsos, C.: Validating policies for dynamic and heterogeneous cloud environments. In: Working Conference on Virtual Enterprises, Springer (2016) 506–517
19. Verginadis, Y., Patiniotakis, I., Mentzas, G., Veloudis, S., Paraskakis, I.: Data distribution and encryption modelling for PaaS-enabled cloud security. In: Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on, IEEE (2016) 497–502
20. Veloudis, S., Paraskakis, I.: Ontological templates for modelling security policies in cloud environments. In: Proceedings of the 20th Pan-Hellenic Conference on Informatics, ACM (2016) 65
21. Veloudis, S., Paraskakis, I.: Defining an ontological framework for modelling policies in cloud environments. In: Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on, IEEE (2016) 277–284
22. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F.: Attribute-based access control. Computer **48** (2015) 85–88
23. Veloudis, S., Verginadis, Y., Patiniotakis, I., Paraskakis, I., Mentzas, G.: Context-aware security models for PaaS-enabled access control. In: Proceedings of the 6th International Conference on Cloud Computing and Services Science-Volume 1 and 2, SCITEPRESS-Science and Technology Publications, Lda (2016) 202–212

24. Verginadis, Y., Mentzas, G., Veloudis, S., Paraskakis, I.: A survey on context security policies in the cloud. In: Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on, IEEE (2015) 589–594
25. Verginadis, Y., Michalas, A., Gouvas, P., Schiefer, G., Hübsch, G., Paraskakis, I.: PaaSword: A holistic data privacy and security by design framework for cloud services. Journal of Grid Computing (2017) 1–16
26. Veloudis, S., Paraskakis, I., Verginadis, Y., Patiniotakis, I., Mentzas, G.: A generic framework for representing context-aware security policies in the cloud. In: International Conference on Cloud Computing and Services Science, Springer (2016) 339–359
27. Gouvas, P., Bouras, T.: Policies enforcement middleware and IDE plugin - Deliverable D3.2. https://www.paasword.eu (2017) Accessed: 2017-09-10.
28. OASIS: extensible Access Control Markup Language (XACML) Version 3.0. http://docs.oasisopen.org/xacml/3.0/xacml3.0corespecosen.html (2013) Accessed: 2017-09-10.
29. Mantzouratos, S., Dimitrakopoulos, G.: PaaSword framework - early release - Deliverable D5.2. https://www.paasword.eu (2016) Accessed: 2017-09-10.
30. 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.R., Xu, L., eds.: Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings. Volume 8128 of Lecture Notes in Computer Science., Springer (2013) 180–193
31. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997, IEEE Computer Society (1997) 394–403
32. Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., Xu, Y.: Two can keep a secret: A distributed architecture for secure database services. CIDR 2005 (2005)
33. Ciriani, V., Di Vimercati, S.D.C., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation and encryption to enforce privacy in data storage. (In: Computer Security–ESORICS 2007)
34. Ciriani, V., Vimercati, S.D.C.D., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Combining fragmentation and encryption to protect privacy in data storage. ACM Transactions on Information and System Security (TISSEC) (2010)
35. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: International Conference on the Theory and Application of Cryptology and Information Security, Springer (2010) 577–594
36. Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., Steiner, M.: Dynamic searchable encryption in very-large databases: Data structures and implementation. In: NDSS. Volume 14. (2014) 23–26
37. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. Cryptology ePrint Archive, Report 2016/718 (2016) http://eprint.iacr.org/2016/718.
38. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security **19** (2011) 895–934
39. Schork, S.T., Schwichtenberg, A., Alexakis, S., Moldovan, G.: Application of the holistic data privacy and security framework PaaSword: A focus on the integration in industrial pilots. In: Proceedings of the 12th International Conference on Availability, Reliability and Security. ARES '17, New York, NY, USA, ACM (2017) 94:1–94:10
40. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order-preserving encryption for numeric data. In: SIGMOD Conference, ACM (2004) 563–574

41. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: EUROCRYPT. Volume 5479 of Lecture Notes in Computer Science., Springer (2009) 224–241
42. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: ACM Conference on Computer and Communications Security, ACM (2015) 644–655
43. Durak, F.B., DuBuisson, T.M., Cash, D.: What else is revealed by order-revealing encryption? In: ACM Conference on Computer and Communications Security, ACM (2016) 1155–1166
44. Kellaris, G., Kollios, G., Nissim, K., O'Neill, A.: Generic attacks on secure outsourced databases. In: CCS. (2016)
45. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, ACM (2009) 169–178
46. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. Journal of the ACM (JACM) **43** (1996) 431–473
47. Lacharit, M.S., Minaud, B., Paterson, K.G.: Improved reconstruction attacks on encrypted data using range query leakage. Cryptology ePrint Archive, Report 2017/701 (2017) http://eprint.iacr.org/2017/701.
48. Stefanov, E., Shi, E., Song, D.: Towards practical oblivious RAM. CoRR **abs/1106.3652** (2011)
49. Bost, R.: Sophos - forward secure searchable encryption. Cryptology ePrint Archive, Report 2016/728 (2016) http://eprint.iacr.org/2016/728.
50. Zhang, Y., Katz, J., Papamanthou, C.: All your queries are belong to us: The power of file-injection attacks on searchable encryption. Cryptology ePrint Archive, Report 2016/172 (2016) http://eprint.iacr.org/2016/172.
51. Stefanov, E., Papamanthou, C., Shi, E.: Practical dynamic searchable encryption with small leakage. In: NDSS. Volume 14. (2014) 23–26
52. Miers, I., Mohassel, P.: Io-dsse: Scaling dynamic searchable encryption to millions of indexes by improving locality. IACR Cryptology ePrint Archive **2016** (2016) 830
53. Gabel, M., Mechler, J.: Secure database outsourcing to the cloud: Side-channels, countermeasures and trusted execution. In: IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS), IEEE (2017)
54. Costan, V., Devadas, S.: Intel SGX explained. Cryptology ePrint Archive, Report 2016/086 (2016) http://eprint.iacr.org/2016/086.
55. Kurosawa, K., Ohtaki, Y.: How to update documents verifiably in searchable symmetric encryption. In: CANS. Volume 8257 of Lecture Notes in Computer Science., Springer (2013) 309–328
56. Bost, R., Fouque, P.A., Pointcheval, D.: Verifiable dynamic symmetric searchable encryption: Optimality and forward security. Cryptology ePrint Archive, Report 2016/062 (2016) http://eprint.iacr.org/2016/062.
57. Xiao, X., Paradkar, A., Thummalapenta, S., Xie, T.: Automated extraction of security policies from natural-language software documents. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. FSE '12, ACM (2012)
58. Slankas, J., Xiao, X., Williams, L., Xie, T.: Relation extraction for inferring access control rules from natural language artifacts. In: Proceedings of the 30th Annual Computer Security Applications Conference. ACSAC '14, ACM (2014)