SSICLOPS

Report on Body of Knowledge in Secure Cloud Data Storage

SSICLOPS Deliverable D2.1 (revision 2)

Felix Eberhardt[2], Jens Hiller[1], Oliver Hohlfeld[1], Stefan Klauck[3], Max Plauth[2], Andreas Polze[2], Matthias Uflacker[3], and Klaus Wehrle[1]

[1] *Communication and Distributed Systems, RWTH Aachen University, Germany*
[2] *Operating Systems and Middleware Group, Hasso Plattner Institute, Germany*
[3] *Enterprise Platform and Integration Concepts Group, Hasso Plattner Institute, Germany*

December 23, 2016

This report has received funding from the European Union's Horizon 2020 research and innovation program 2014–2018 under grant agreement No. 644866 ("SSICLOPS"). It reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

Contents

1. Introduction

The need for increased awareness regarding privacy and security is more urgent than ever for many reasons. The rising number of data leaks of sensitive customer data or corporate trade secrets is a major concern which companies have to deal with when they want to expand their businesses leveraging cloud resources.

Antagonizing the abundance of security threats faced in the Internet does not only help to increase the overall level of security of cloud resources, but it also helps to increase the perceived level of trustworthiness associated with cloud computing. Hence, the goal of this document is to provide a body of knowledge about enhanced privacy and security concepts which facilitate secure cloud computing infrastructures.

As a first intermediate result, this document provides a comprehensive overview of the current state of the art regarding *policy languages* and *security-aware storage and processing*.

2. State of the Art: Policy languages

Originally, *policies* described advices, restrictions or obligations on the actions of humans, e.g., legal rules describe policies for the interaction in the human society. When the Internet enabled the data exchange of humans with servers, it also became necessary to specify policies for the handling of data that arises during this communication such as IP addresses. Moreover, Internet Service Providers are able to track which websites a customer requests, operators of news-websites can track interests of visitors, and email-providers have access to emails of their customers. Typically, companies communicate the handling of such data to the users and their employees in policy statements that use human language but mainly consist of legal phrases. However, companies also must ensure that their employees and systems adhere to specified policies. Automated control systems can help to enforce this adherence to a policy, e.g., an access control system can prevent that an employee accesses personal emails of a customer. However, automated systems are not able to understand policies that are specified in human language, which is the format used to communicate them to customers and employees.

To overcome this limitation, *policy languages* define a machine-readable format for the representation of a policy in order to enable its automated processing. Early policy languages focused on mere access control, i.e., they allowed specifying which individual or system obtains access to (sensitive) data. The application of such access control languages is not limited to email providers or providers of websites, as outlined above, but also used for the access control when processing personal information, e.g., in the area of accounting, banking, handling of insurance information or processing of medical information of patients. Furthermore, they cannot only specify the access to data of employees and own systems, but also control the access to data by third parties, e.g., subsidiaries or collaborating marketing companies.

However, as we present in the following, cloud computing demands for a policy language that provides expressiveness beyond the boundary of access control. To this end, we explain the use case of policy languages in the scenario of cloud computing and derive requirements for a policy language tailored to the cloud in Section 2.1. Following up, we present the state of the art regarding policy languages and their support of the derived requirements for a cloud-focused policy language in Section 2.2. In Section 2.3, we discuss the derived requirements with the goal to guide further development of a policy language that is specifically tailored to the cloud. A description of related work in Section 2.4 finished the part on state of the art of policy languages.

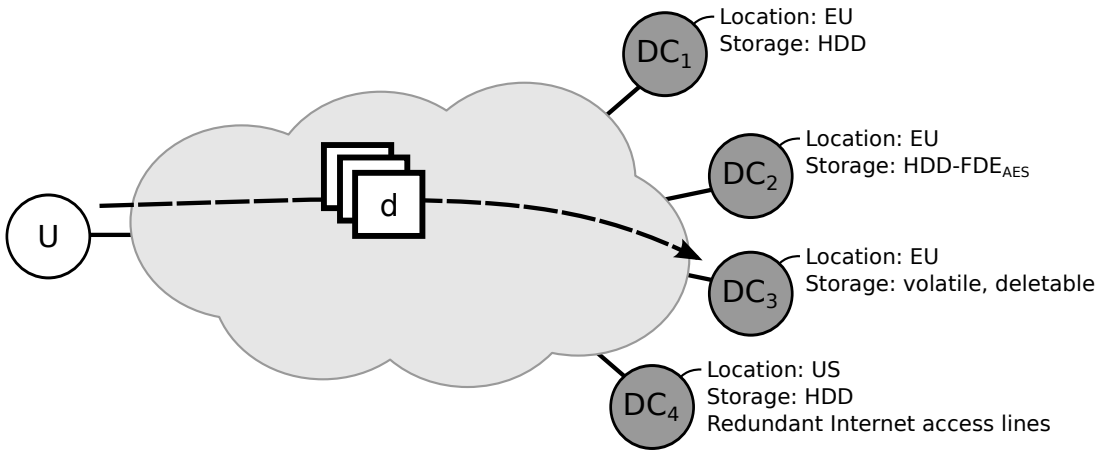

Figure 1: Scenario for privacy policy languages in cloud computing. The user (U) stores and processes data (d) with the help of cloud resources. The depicted cloud data centers DC_1, DC_2, DC_3, and DC_4 are located in the European Union (EU) or in the United States (US). They store data on standard storage hardware (HDD), employ full disk encryption based on AES (HDD-FDE_{AES}) or provide volatile storage that allows reliable data deletion.

2.1. Cloud Scenario

The growing number of cloud services and the shift towards processing data in the cloud, instead of performing computations locally, results in an increasing amount of personal data being stored and processed by cloud services. With an increasing awareness regarding privacy of their data, users of cloud services expect cloud operators to handle their data with respect to their privacy, e.g., users demand that cloud providers store sensitive user data encrypted, or guarantee deletion of data when it is no longer required [27].

Notably, the preferences regarding privacy may vary depending on the privacy-awareness of the specific user and the offer of cloud services. Exemplarily, Figure 1 depicts a user that stores personal data with the help of a cloud service for file storage. The cloud service uses different data centers to provide this service. The data centers are located at different continents, employ different storage hardware and provide additional features such as an increased availability, e.g., guaranteed with the help of redundant Internet connection. The selected data center may vary on the preferences of the actual user, e.g., a citizen of the European Union (EU) may wish to benefit from more strict privacy regulation by storing data in one of the data centers located in the EU, while a citizen of the US may wish to benefit from faster access to data achieved when storing it in his/her country. Furthermore, preferences may not only vary between different users, but also diverge for a single user depending on the sensitivity of data, e.g., a user may wish to wipe out data that corresponds to a loan as soon as reimbursement is completed, while comments on a newspaper article are less sensitive and may not be deleted at all.

In the small example depicted in Figure 1 the user could select the suitable data center on his/her own. However, more complex policies as well as advanced management techniques in the cloud such as replication and load balancing require that cloud services become agnostic to the policy of users in order to automatically process and adhere to user demands. Thus, a cloud service that accounts for the diversity of demands can no longer rely on a single service-wide policy as different data must be handled according to specific policies. Instead of a single static policy, a cloud service could offer users the ability to choose among several policy profiles. However, if users specify the policy on their own and there exists a language that is supported by different cloud services, users would, first, be able to specify a policy exactly tailored to their demands and, second, could use this policy for different cloud services without the need to choose among possibly non-standardized policy profiles. Hence, it is reasonable to enable users to specify a policy, which is specifically tailored to their demands, themselves and submit this policy to cloud services. Still, a cloud service has the ability to reject a user-specified policy and, thus, can select which policies it supports.

The example shows that the main purpose for policies in the scenario of cloud computing is the specification of rules that regulate the privacy of user data. Thus, the scenario demands for a *privacy policy language*, i.e., a policy language that focuses on the expression of privacy related rules. Especially, a privacy policy language for the cloud requires rules beyond the limits of access control. Notably, none of the depicted features of the data centers belongs to the area of access control. Thus, a policy language that is limited to the specification of access control rights, which was the purpose of early policy languages, does not provide the required expressiveness to cope with policies required in the cloud computing scenario. In order to provide a suitable basis for the discussion of the state of the art of policy languages, we first derive the functional requirements for a privacy policy language in the area of cloud computing.

2.1.1. Requirements

Before we present the state of the art of policy languages, we derive the requirements of a policy language for the scenario of cloud computing. This enables us to judge existing (privacy) policy language with respect to their applicability in cloud computing and to extract existing mechanisms to fulfill specified requirements.

2.1.1.1. Storage footprint

In the cloud scenario, users specify policies for their data items that are stored and processed by cloud services. Before a cloud service can use a data item, it must check if the corresponding policy allows it to execute the desired action. Thus, cloud services require access to policies of data items that they store or process. Notably, cloud services often perform actions when the user is not connected to the service, e.g., indexing, data replication, or long running jobs. In order to comply with the policy a user specified for a data item that is affected by such a background task, the service would need to evaluate the policy without the user being present.

Hence, assuming that only the user stores policies for his/her files, a cloud service has to delay background tasks until it can request required policies from the user. However, this decreases the usability and efficiency of cloud services, e.g., moving and replicating data ensures a suitable load balancing for cloud resources and guarantees availability, respectively. Alternatively, users can store policies at a service that provides high availability such that cloud services can retrieve policies even when they are absent, e.g., the cloud service could store relevant policies itself or users could store them at another cloud service.

In either case, the overhead depends on the storage footprint of a policy. If the cloud service does not store policies itself, i.e., the policies remain with the user or are offloaded to another cloud service, policies must often travel through the network. In this case, a small storage footprint decreases the network overhead introduced by the support of the privacy policy language. Otherwise, if policies are stored by the cloud service itself, the storage footprint of policies determines the storage overhead and, thus, should be small. Notably, delegating the storage of policies to a second cloud service causes both overheads, additional network traffic and an increased demand of storage resources.

Thus, a privacy policy language for the cloud must represent policies with a *small memory footprint*.

2.1.1.2. Readability

The privacy policies described in Section 2.1 represent the policy of a user regarding the handling of his/her data by cloud services. Users may either specify the policies on their own or obtain and use a policy from a trusted third party [27], e.g., the office of fair trading, easily understandable generators, or other experts in the field. In either case, users should be able to understand the meaning of the policy, which requires them to be able to read the policy in the first place. Thus, policies of a privacy policy language for the cloud scenario must be *readable by humans*.

Additionally, cloud services must process the user-specified policies, which requires the policies to be *readable by machines*. Notably, this requirement prevents the usage of general human language as current techniques cannot reliably interpret it.

2.1.1.3. Comprehensibility

As already stated in the previous requirement, readability by humans is required to understand the meaning of a given policy. Users should be able to determine the meaning of a policy, i.e., the policy should be *comprehensible*, in order to enable users to compare a policy with their demands. This also enables them to use policies received by third parties as a baseline and adapt them to their special needs.

Furthermore, the confidence of users in the privacy policy language can benefit if the users are able to understand the evaluation process that determines if a cloud service or specific resource is

able to comply with a user's policy. Thus, a privacy policy language should use a *comprehensible evaluation process* that can also convince non-expert users.

2.1.1.4. Conflict detection or conflict handling

As already elaborated above, cloud services perform actions with data of users when the user is not connected to the cloud. This becomes a problem if a policy contains a conflict, e.g., if one rule of the policy specifies that privacy sensitive data must be stored in the EU, but another rule states that medical data, which is privacy sensitive, must be stored by a specific data center in the US. In this case, the cloud service cannot perform the action without enquiring the user, who is possibly unavailable such that the task must be aborted. Thus, a privacy policy language for the cloud must prevent the occurrence of conflicts in order to not hinder background tasks of cloud services.

One possibility to prevent conflicts is a check for conflicts during the specification of the policy. The user can obtain direct feedback regarding the problem and resolve the conflict based on his/her preferences. Hence, a privacy policy language should support the *detection of conflicts during policy specification*.

If a privacy policy language does not detect conflicts during specification, it still may provide solutions to cope with conflicts, e.g., always use the most restrictive rule that matches and provide a preference order, e.g., for countries. Thus, an alternative to conflict detection at time of specification are *conflict handling mechanisms*.

2.1.1.5. Performance

The policy of a data item must be evaluated each time an action could be restricted by the policy. Thus, cloud services have to perform evaluation of policies on a frequent basis. In order to limit the corresponding overhead, a privacy policy language for the cloud must come with a good *evaluation performance*.

Depending on the solution to provide human and machine readability, specifically if a privacy policy languages uses different representation to encompass for both needs, the *performance of the transition mechanism* between these representations may also be of concern.

2.1.1.6. Expressiveness

As discussed earlier, the cloud scenario depicted in Figure 1 already showed that a privacy policy language cannot be limited to the specification of access control rights. Instead, it requires extended expressiveness.

While a privacy policy language also must support the specification of access rights, e.g., to control the access by subsidiaries or third parties, a user must also be able to specify *obligations*,

i.e., rules that specify actions that the cloud service has to perform. An example for the usage of obligations is the preference of a user that information regarding a loan are deleted after reimbursement. Other preferences specified with the help of obligations are (i) restriction of storage location to a certain continent or country, (ii) deletion of a data item at a specified point in time, (iii) logging or notification when data is accessed by a third party, or (iv) replication rate of data to ensure availability. Thus, obligations are a key requirement of a policy language for the scenario of cloud computing.

The third example also shows that the privacy policy language must support *event-based triggers* in order to specify the condition under that the data of the loan must be deleted. Similarly, the language must support *time-based triggers*, e.g., to allow a user to specify an obligation that results in deletion of a data item after a certain time period, e.g., to satisfy legal retention periods but decrease storage load as soon as possible.

Additionally, a privacy policy language for the cloud must be agnostic to the *environmental context* of the resources used by the cloud service, e.g., to determine the location of a data center such that a user can specify the storage location. This enables a much more fine granular specification of the storage location by the user compared to the prior state where users could only choose among cloud services. Other examples that require the language to recognize the environmental context are storage methods, encryption ciphers, integrity protection modes, replication factor, or network connectivity.

2.1.1.7. Extensibility

The last years showed a rapid development in the area of cloud computing with cloud services offering previously not covered services coming up. New cloud services may require users to adapt their policies to these services, which can require the support of new functionalities by the privacy policy language. Thus, a privacy policy language for the cloud must be *extensible* in order to adapt to emerging cloud services that require new functionality.

2.1.1.8. Enforcement

Up to now, we have not considered the case that the cloud provider deviates from a user-specified policy. Without techniques that monitor the behavior of the cloud service, users have to trust the cloud operator to adhere to submitted policies or reject policies the service cannot or does not want to fulfill. Furthermore, users must trust that cloud services are protected against attacks of external attackers. However, users may not trust the operator to properly fulfill policies and protect the system against attacks. For these users, we can build up trust into the system by technically enforcing the adherence to policies. Thus, support of *policy enforcement mechanisms* is beneficial for a cloud privacy policy language.

Additionally, the enforcement of policies can be beneficial if third parties are involved, e.g., when a hospital uses a cloud service to process data. In this case, the user submits his/her policy to the

hospital that relays data and the policy to the third party cloud service that still has to adhere to the policy. If the hospital cannot enforce the adherence to the policy by third party cloud services, it puts its reputation at risk as the hospital may be blamed for the selection of a third party service that deviates from the policy.

2.1.1.9. Development

The support of a privacy policy language by cloud services involves new duties for the development of cloud services. Cloud services must evaluate and act according to the rules specified in the policy. However, most developers are not experts in the area of privacy and policies. Hence, a privacy policy language requires a comprehensible abstraction level that developers can use to, e.g., get information if the service is allowed to process or share a data item.

2.1.1.10. Matching

We already brought up that cloud services may reject a policy that they cannot or are not willing to fulfill, e.g., a cloud service that exclusively uses data centers located in the US cannot satisfy a policy that forces usage of data centers located in the EU. In order to not fall back to certain policy profiles that a cloud service is willing to accept, which would limit the flexibility users gain by specifying their policies on their own, we require an automated process that checks if a user-specified policy complies with the demands of a cloud service [27]. To this end, the provider of a cloud service can specify its rules in a policy, similar to the policy specified by the user. In order to obtain a result, a privacy policy language must be able to compare both policies and determine if they *match*, i.e., the cloud service can fulfill the policy, or do not match, i.e., the cloud service must reject the request of the user based on the policies.

2.2. State of the Art

In this section we present the state of the art of (privacy) policy languages with respect to their applicability in the cloud scenario. Thereby, we build upon the requirements for a privacy policy language in the scenario of cloud computing elaborated in Section 2.1.1 and describe which mechanisms existing (privacy) policy languages employ to achieve them.

2.2.1. XACML

The *eXtensible Access Control Markup Language* (XACML) [24] is the predominant language for the specification of access control rights.

As many other languages that we cover in the following, XACML relies on XML to represent policies. XML provides a reasonable storage footprint, especially when compression is applied

before storage. However, we cover more concise representations that use optimizations at the level of bytes later on. Nevertheless, representing policies as XML enables humans to read this representation, but understanding this format is still complicated for users and, thus, only developers and other experts benefit from this approach. A benefit of XML is its extensibility such that XACML can, with respect to the representation, be easily extended when new requirements on the language emerge.

With respect to the requirements for a cloud privacy policy language, XACML lacks desired features. It is not agnostic to conflicts and its does not support triggers. Admittedly, XACML is designed to serve as a building block that can adapted to the desired use case, e.g., to build up a privacy policy language tailored for the demands in the cloud computing scenario. However, this requires extensions to support conflict detection and triggers as these are required to fit the scenario of cloud computing. Still, such an extended version would use XML and at least require an additional abstraction to present policies to users in an easily readable and comprehensible format.

2.2.2. PERFORM

The *PERvasive FORmal privacy language* (PERFORM) [11] targets the scenario of pervasive computing [67] with the help of a human readable language that employs a non-expert understandable evaluation process. Notably, the scenario of pervasive computing and the scenario of cloud computing are similar with respect to a high amount of machine to machine communication as well as interaction with users and processing of privacy sensitive data. Like XACML, PERFORM relies on XML without further abstraction layer to provide human readability.

Policies in PERFORM specify actions as request/response pairs and limit these with the help of constraints. Before the desired action, i.e., the response, is executed, the system checks conditions that are given in the request and determines if the constraints allow the response. Thereby, PERFORM employs an evaluation mechanism tailored to be understandable for non-experts. To this end, it aligns the evaluation with respect to the steps a non-expert would use for evaluation [11]. Specifically, the evaluation engine (i) sets up the environmental context, (ii) evaluates constraints that depend on the determined context, (iii) evaluates constraints that check the actual request, and (iv) executes the response or reports that the desire does not conform with the policy.

With respect to contradicting rules, PERFORM does not support the user in detecting conflicts and rejects a response if any condition of the request is not met or any constraint forbids the action [11], i.e., a response is not carried out in case of contradicting statements. Concerning the expressiveness, PERFORM is agnostic to the environmental context and allows the notification of users as well as logging of performed actions [11]. However, it does not support triggers. Hence, actions must be triggered by users and cannot be based on timestamps or user-independent events.

2.2.3. Rei

Rei [34] is a policy language that, similar to PERFORM (cf. Section 2.2.2), was developed for the application in pervasive computing environments. In contrast to PERFORM, Rei is implemented in Prolog.

Concerning expressiveness, Rei supports the specification of rights, prohibitions, obligations, and dispensations. Furthermore, policies can depend on the environmental context. However, Rei does not support triggers.

A special feature of Rei is the ability to resolve conflicts. As Rei supports rights, prohibitions, obligations, and dispensations, conflicts can occur between rights and prohibitions, between obligations and dispensations, as well as between obligations and prohibitions [34]. In order to resolve these conflicts, Rei introduces *meta-policies*, which specify priorities or precedence relations. For two rules, a priority specifies which rule is applied in case that the rules are conflicting. Similarly, a user can set up priorities among policies (sets of rules), e.g., a security policy may always override a load balancing policy. In addition to priorities, a user can specify precedence relations, which allow for the specification of default rules to resolve conflict. These precedence relations have either a positive or a negative modality. In case of negative modality, prohibitions overwrite rights, dispensations hold over obligations, and prohibitions overrule obligations, while positive modality has the opposite effect [34]. Thereby, users can specify constraints that restrict a precedence relation to certain actions or entities [34], e.g., a family member may get file-access in case of a conflict, while a colleague does not.

Furthermore, Rei enables a user to restrict actions (i) to occur in a predefined sequence, (ii) to not occur with a specified other action, or (iii) to be performed repeatedly or only once [34]. Thus, it enables users to build up a policy from small rules that are used as building blocks for large policies.

2.2.4. *Garcia-Morchon et al.*

Garcia-Morchon et al. [21] propose an access control policy language for medical sensor networks. The resource constraints in this environment demand for a very concise representation of policies such that they can be stored on the small amount of memory provided by sensors. To this end, Garcia-Morchon et al. specify policies in boolean formulas that are represented as binary trees [21]. A node within this tree represents an atom, e.g., a condition, and the left-hand child is connected to this condition with a logical AND operator, while the right-hand child is connected with a logical OR operator. In order to efficiently encode this trees, Garcia-Morchon et al. use a byte-level encoding. While the root node is represented by 2 bytes that specify the length of the tree, each other node is encoded with a length and a value field. The value field contains an encoding of the condition that the node represents. Furthermore, children and their descendants are encoded appending the encoded subtree to the value field. In order to obtain a concise

encoding of the value fields, it uses predefined byte sequenced, e.g., operators are encoded with a single byte: 0x00 ($=$), 0x01 ($\neq$), 0x02 ($\leq$), 0x03 ($<$), 0x04 ($\geq$), and 0x05 ($>$).

Notably, the described encoding of trees is not readable for humans. Hence, Garcia-Morchon et al. use an abstraction from the encoded tree when a policy is presented to humans.

With respect to contradicting rules, the language avoids conflicts with the help of a strict order and applies the first matching rule, i.e., the first matching rule determines the decision outcome [21]. Notably, this approach requires that rules are placed in front of other rules that have a higher degree of generality such that general rules do not shadow more specific rules [21]. The duty to ensure an order that fulfills this condition is left to the user.

2.2.5. *Ali et al.*

Ali et al. [1] describe an obligation language and a framework to enable privacy-aware service oriented architectures. Their work bases upon work done in the PrimeLife project[1]

Their language supports the specification of obligations, can evaluate the environmental context and supports time- as well as event-based triggers. Thus, expressiveness of their language matches the requirements that we derived in Section 2.1.1.6.

2.2.6. OSL

The *Obligation Specification Language* (OSL) [29] is a policy language for distributed usage control. In contrast to the other policy languages, OSL partially supports the enforcement of policies. However, OSL does not specify a new enforcement mechanism tailored to policy languages. Instead, it employs the enforcement mechanisms of digital rights management (DRM) languages. To this end, Hilty et al. show that a subset of OSL can be translated into the DRM language *Open Digital Rights Language* (ODRL). Thus, the obligations of restricted OSL policies can be enforced with the mechanism provided by ODRL[29]. Similarly, a subset of OSL can be translated into the *eXtensible Rights Markup Language* (XrML) and enforced with XrML enforcement mechanisms [29].

2.2.7. C^2L

C^2L [58] allows the specification of policies that restrict the location and migration of virtual machines (VMs) in the scenario of cloud computing. This enables a company, e.g., to specify that their VMs must not run on the same server as VMs of competitors in order to prevent side-channel attacks. Notably, this use case belongs to cloud computing but is different from the

[1] http://primelife.ercim.eu/

cloud scenario we focus on as we aim for privacy of user data in general and do not restrict to policies for the management of VMs (cf. Section 2.1).

Notably, C^2L enables enforcement of policies if the history of the placement and migration of VMs is reliably stored. Specifically, a user can verify if a given history contradicts against a policy specified in C^2L. As C^2L policies are specified in a typed spatio-temporal logic, it suffices to rerun the evaluation engine with the given history. Notably, all temporal operators express constraints on the past, i.e., operator can specify that a condition (i) holds on some past state, (ii) holds in the previous state, or (iii) holds in all past states. Thus, a user can, assuming a reliable source for the history of VM placement, a posteriori verify the adherence to the specified policy.

2.2.8. S4P

S4P [5] focuses on the matching of user-specified privacy policies and privacy policies of service providers. Before providing personal identifiable information to a service, a user has to *permit* the actions defined by the provider while the provider has to *promise* to adhere to the policy specified by the user. To this end, S4P policies are specified in first-order function-less signatures and the policy of a user is compared with the policy of a service provider using formal mechanisms for this formal logic.

Notably, the first-order function-less signatures as used by S4P are still human readable as they consist of horn logic fragments that are expressible as if-statements [5]. Hence, an expert can easily read and understand policies. However, the evaluation engine uses formalisms similar to, e.g., first order logic. Thus, it requires a deeper knowledge of logic and is not suitable for non-experts.

2.2.9. FLAVOR

FLAVOR [63] is a policy language that allows a posteriori verification for the scenario of legal rules. Notably, the approach used in FLAVOR goes beyond the simple check whether a service adhered to a policy as possible with C^2L policies when employing VM placement histories (cf. 2.2.7). Instead, FLAVOR does not only specify which policy a system should adhere to, but which actions have to be taken if a rule is breached. These a posteriori checks overcome the limits of a priori checks that are constraining and often not enforceable [63]. To this end, *Contrary-to-duty obligations* define the behavior in case of deviation from the main policy rules.

Concerning expressiveness, FLAVOR combines contrary-to-duty obligations with support of temporal as well as deontic modalities in order to express legal rules. Thereby, temporal modalities enable the specification of obligations with deadlines [63]. Furthermore, FLAVOR

uses a trigger feature to integrate external events into rules [63]. Finally, obligations can base on context information. Thus, FLAVOR provides the expressiveness derived in Section 2.1.1.6.

2.3. Discussion

In the previous section, we presented the state of the art of policy languages with respect to the requirements that we derived for the scenario of cloud computing (cf. Section 2.1.1). Based on the obtained knowledge of the already existing mechanisms that tackle the corresponding challenges, we now discuss the derived requirements. Specifically, we classify them according to their importance in the scenario of cloud computing and, when required, point out approaches for new mechanisms in order to obtain a guideline for the development of a privacy policy language that is specifically tailored for the needs of the cloud computing scenario. Following up, we give an overview on the support of the cloud scenario by the already covered existing privacy policy languages.

2.3.1. Discussion of requirements

In Section 2.1.1, we listed requirements for a privacy policy language in the scenario of cloud computing, but neither derived their relative importance, nor discussed their interoperability as this requires knowledge of the mechanisms that tackle the challenges. In the following, we discuss the importance of each requirement in order to come up with a guideline for the development of a privacy policy language for the cloud computing scenario. Furthermore, we discuss the existing mechanisms presented in Section 2.2 and outline new mechanisms if suitable methods are missing.

2.3.1.1. Storage footprint and location

In Section 2.1.1.1 we already listed the possibilities to either (i) retrieve policies from the user, (ii) store and retrieve policies with the help of a third party, e.g., another cloud service, or (iii) store policies at the cloud service itself.

In the first two cases, each usage of a data item requires transmission of the policy over the network for the purpose of policy evaluation. In contrast, the storage by the cloud service itself only requires local communication as long as the data item and its policy are stored at the same data center. Hence, we propose a strong coupling of data and policy for data that lives in the cloud. Specifically, a policy should always travel along with its data item as *data annotation* [27]. In the optimal case, data and policy are not only stored in the same data center but also at the same physical storage medium.

While the approach to use data annotations limits the networking overhead, data annotations still increase the size of the initial transmission, i.e., when data travels to the cloud, and require

storage capacity at the data center. Hence, a small storage footprint is still required to decrease the storage requirements for cloud services and keep additional intra-data-center traffic for evaluation of policies reasonable. In order to decrease the storage footprint, a privacy policy language for the cloud should employ a representation that enables optimization on the byte or bit-level similar as proposed by Garcia-Morchon et al. [21]. Furthermore, policies may be checked for tautologies to cut off irrelevant subsets of a policy and, thus, decrease the size of a data annotation. Finally, a cloud service may also employ mechanisms to find redundancies among policies and employ profiles for often occurring combinations to further reduce the size of data annotations.

2.3.1.2. Readability with respect to storage footprint

We derived the requirement that a policy must be readable by humans as well as by machines (cf. Section 2.1.1.2). Most of the policy languages described in Section 2.2 represent policies in XML to achieve human readability as well as machine readability. However, with respect to the requirement of a small storage footprint, Garcia-Morchon et al. use a representation of policies that is optimized on the byte-level [21]. Such optimizations on the level of bytes, or even on the bit-level, are not supported by XML. Thus, using XML is not the optimal solution to achieve the requirement of a small storage footprint.

Consequently, a privacy policy language for the cloud must tackle the challenge to use a representation that is highly optimized with respect to the storage footprint but is still readable by humans. To this end, it could provide two representations of a policy, i.e., one for the human and one for the machine domain [21]. Specifically, one representation can be dedicated to the special requirements of humans such that users can easily read and understand the policy. This opens the possibility to even abstract from the XML syntax and present policies to humans in a better understandable format. Then, the other representation that is used by machines can account for the requirement of a small storage footprint without the constraint to be readable by humans. As this approach enables optimizations on the level of bits, it is much more promising than the usage of a single representation for both domains. Notably, except for the language of Garcia et al. that is very specifically tailored for medical sensor networks and stops optimization at the level of bytes instead of bits, none of the presented languages leverages this possibility.

2.3.1.3. Comprehensibility

If a policy is not comprehensible, users are not able to check if a policy suites their demands. The comprehensibility of a policy can be increased with a suitable representation. Specifically, a policy represented as XML is readable by humans, but not as comprehensible as a written human language or a representation as flow chart.

Regarding the comprehensibility of the evaluation process, the trust of users in the privacy policy language may increase if they can understand the process of policy evaluation. However, a comprehensible evaluation engine as proposed by PERFORM may have negative impact on the

performance of the evaluation process and thus decrease quality of experience and increase costs. In order to prioritize performance, trusted auditors can certify the correctness of the evaluation engine. Hence, while a privacy policy language for the cloud benefits from a comprehensible evaluation engine, there are alternatives to build up trust of users in this process without possibly limiting the performance of the evaluation engine.

2.3.1.4. Conflict detection or conflict handling

We already elaborated the benefit of conflict detection or conflict handling in Section 2.1.1.4. Considering that conflict detection raises a notification and guides the user to resolve the conflict, a privacy policy language can cope with conflicts either by conflict detection, i.e., detect a conflict at point of specification, notify the user, and guide him/her to a solution, or with the help of conflict handling, i.e., handling conflicts at runtime.

Conflict handling mechanisms, e.g., using meta-policies to specify hierarchies among sub-policies of a policy [34] or using an order among rules and apply the first matching one [21], hide specific conflicts from the user. Hence, the user does not have to understand and resolve a specific conflict. However, this comes at the cost of a less specific policy definition that may defer from the users requirements, e.g., the hierarchy of a security sub-policy and a medical sub-policy may depend on the current medical context [21]. Similarly, the application of the first matching rule requires careful ordering of rules as general rules can shadow more specific rules [21]. This shadowing can remain unnoticed by the user without automated checking and corresponding notification.

In contrast, applying conflict detection when a user specifies his/her policy enables notification of the user who can take a decision tailored to the specific conflict. Furthermore, the evaluation engine can assume the absence of conflicts which may lead to a simpler and faster evaluation process.

2.3.1.5. Performance

The performance of the evaluation of a policy affects the overall performance of cloud services. In order to not negatively affect the operation of cloud services, especially with respect to *Quality of Service* and *Quality of Experience*, this requirement is of utmost importance. If the overhead that the privacy policy language introduces exceeds reasonable limits, service providers may even refuse to support it.

Furthermore, the performance of the evaluation engine becomes even more important if not only cloud services that employ resources of data centers evaluate policies. A privacy policy language for the cloud may also allow users to specify rules that require evaluation at less powerful middleboxes such as routers, e.g., a policy could specify that a data item should only be routed within a certain country. Hence, the evaluation engine must be efficient enough to also be applicable on the comparable less powerful hardware of middleboxes.

Finally, we pointed out the solution to use two representations of a policy to account for a small storage footprint, readability, and comprehensibility. Due to the small storage footprint we assume that the machine representation becomes the standard format of a policy. The human readable format is only required by the user to specify and check a policy, i.e., the cloud service never makes use of the human readable representation. Hence, an efficient translation for the machine domain representation to the human domain is less important.

2.3.1.6. Expressiveness and extensibility

The expressiveness is of utmost importance for a privacy policy language for the cloud. Users can benefit from a standardized language that is supported by most cloud services such that users leverage synergies by reusing the same policies among different cloud services. However, in order to be suitable for standardization and usage by most cloud services, the privacy policy language must cope with the broad range of required policy rules that comes with the diversity of cloud services.

Specifically, a privacy policy language for the cloud must support (i) obligations, (ii) time-based triggers, (iii) event-based triggers, and (iv) be agnostic to environmental context information (cf. Section 2.1.1.6).

The extensibility is of similar importance for a cloud privacy policy language. Especially with the goal of standardization to enable synergies for users, the language must be able to adapt to emerging cloud services with new requirements regarding the specification of policies.

2.3.1.7. Enforcement

The enforcement of policies with the help of trusted computing technology [9, 36], e.g., trusted platform modules, or with the help of external auditors that frequently check and certify cloud systems [33] can increase the trust of users in cloud services with respect to their adherence to user-specified policies [27]. However, already today users trust that service providers adhere to service level agreements. To this end, legal rules as well as the negative consequences resulting from a breach that gets public provide a sufficient incentive for service providers to not deviate from the expected behavior. Hence, enforcement of policies with technical mechanisms may be beneficial but is not mandatory in the first place. Additionally, enforcement mechanisms increase the overhead (automated mechanisms) or cost (external auditors) and, thus, may decrease the number of cloud services that support a privacy policy language that makes these mechanisms mandatory.

2.3.1.8. Development

A strict decoupling of policy functionality and service functionality would be beneficial for developers [72]. However, while access control can be achieved with an API that automatically

Table 1.: Comparison of policy languages. A language fulfills (+), partially fulfills (~), or does not fulfill (-) a requirement. Some languages do not provide sufficient information to rate all requirements (·). Important requirements (cf. Section 2.3.1) are written in bold.

	storage footprint	**readability**	comprehensibility	**conflict detection**	conflict handling	**performance**	**expressiveness**	**extensibility**	enforcement	development	**matching**
XACML	~	~	~	-	-	·	-	+	-	~	·
PERFORM	~	~	+	-	-	·	~	·	-	·	·
Rei	·	·	~	-	+	·	~	+	-	·	·
Garcia et al.	+	+	+	-	~	·	~	·	-	·	-
Ali et al.	·	·	-	-	-	·	+	·	-	·	·
OSL	·	-	-	-	-	·	~	·	~	·	+
C^2L	~	-	~	-	-	~	~	·	~	·	-
S4P	·	+	-	-	-	·	·	·	-	·	+
FLAVOR	·	-	-	-	-	·	+	·	~	·	-

delivers a placeholder when access rights are not sufficient, without that the developer of the core functionality has to determine if a placeholder or the actual data item must be delivered [72], it becomes hard to abstract from the usage of obligations and triggers as they explicitly have to be maintained, checked and performed by the cloud service. Furthermore, if the privacy policy language for the cloud is standardized, the overhead for developers is limited and a sufficient amount of guidance should be available.

2.3.1.9. Matching

While we mainly focus on the specification of policies by users, the flexibility of user-specified policies requires cloud services to check if their own policy matches with the user's policy. This is especially important as a cloud service must be able to reject data items that require a policy the service cannot adhere to. However, similar as the evaluation engine, the matching process requires a good performance. This performance may benefit if the matching process is directly integrated into the evaluation engine of the policy. More specifically, a cloud service could specify the features that it is willing to provide, e.g., storing data only in data centers located in the EU, and the evaluation engine checks if the provided features are sufficient to conform to the policy.

2.3.2. Discussion of State of the Art

We now provide a concluding overview on the state of the art of policy languages with respect to the requirements we derived for the scenario of cloud computing.

Table 1 shows the results of our analysis of existing (privacy) policy languages. It lists the analyzed policy languages and classifies their support of the different requirements for the cloud scenario.

The table shows that the important requirement of a small storage footprint is so far neglected by almost all privacy policy languages. Similarly, only a few account for a suitable *readability* with the help of abstractions and leave users with analysis of a formal representation. The *comprehensibility* strongly depends on the employed logic, i.e., languages that express policies in formal logic suffer from the complicated evaluation engine. However, using established evaluation engines may result in a better performance, which is not possible to evaluate given the available information. Further, no language explicitly describes support of *conflict detection*. Instead, some languages handle conflicts invisible for the user, which possibly leads to a mismatch between the preferences of a user and the specified policy. Also, no language provides decent information regarding the *performance* of its evaluation engine, which is of importance in the cloud scenario as the application of a privacy policy must not overburden available resources. Comparing the *expressiveness* of the languages, most fulfill the requirements for cloud computing at least partially, but a standardized privacy policy language for the scenario of cloud computing must support all the demanded expressions. Furthermore, the requirement to be *extensible* is only fulfilled by few languages and not sufficiently described for most languages. Our discussion showed that the *enforcement* of policies is not the primary requirement for a privacy policy language. Nevertheless, some languages presented approaches that may be suitable to achieve this requirement in the second place. The support of *development* that abstracts from the privacy policy language is also neglected by most languages, but as our discussion showed, this issue can be addressed by standardization instead of making it a feature of the language. Finally, only a few language account for the requirement to *match* user-policies and service-policies.

Summing up, each language neglects important requirements for the scenario of cloud computing described in Section 2.1. Thus, the state of the art of policy languages does not provide a suitable solution for the scenario of cloud computing.

2.4. Related Work

Kumaraguru et al. [37] presented a survey of privacy policy languages and outline the respective usage scenarios. They categorize the presented languages into (i) sophisticated access control languages (SACL), (ii) web privacy policy languages, (iii) enterprise privacy policy languages, and (iv) context sensitive languages. Furthermore, they derived that some languages are suitable for the representation of user-policies, while other languages are the desired choice to specify enterprise policies. However, Kumaraguru et al. do not cover the usage scenario of cloud computing. Thus, they neither derive requirements for privacy policy languages in the cloud, nor discuss the presented languages with respect to their application in this area.

Ali et al. [1] provide a list of requirements for an obligation-supporting policy language and a policy enforcement framework. Specifically, they outline the importance of (i) independence

of enforcement framework from policy languages, (ii) independence of obligations from data storage, (iii) independence of the framework from communication protocols, (iv) support for common obligations, time-based triggers, and event-based triggers, (v) support for domain specific obligations by the framework, (vi) support for abstraction of actions, (vii) support for abstraction of triggers, (viii) support for distributed deployment of the framework, (ix) support for different trust models, (x) transparency of data handling, and (xi) support for preventive obligations. While (iv), (v), (viii), (ix), and (xi) are similar to our derived requirements, Ali et al. do not show their importance for privacy policy languages in the scenario of cloud computing and do not discuss them in this context. Furthermore, our discussion showed that a privacy policy language for the cloud can benefit if it supports (i) and (x), but that these features are subordinated compared to performance or can be achieved by legislation, respectively.

3. Security-Aware Storage and Processing

Providing low total cost of ownership, high degrees of scalability and ubiquitous access, cloud computing offers a compelling list of favorable features to both businesses and consumers. At the same time, these positive qualities also come with the less favorable drawback, that guaranteeing data confidentiality in cloud-based storage and processing services still remains an insufficiently tackled problem. As a consequence, many companies and public institutions are still refraining from moving storage or processing tasks into the domain of cloud computing. While this reluctance might be appropriate for few, highly sensitive use-cases, it poses the risk of an economic disadvantage in many other scenarios.

This section provides an overview about the current state of the art in security-aware storage and processing. Since there are numerous ways to approach the topic, a large variety of potential starting points is presented. The goal is to provide a solid body of knowledge, which will be used as a foundation upon which novel security mechanisms can be identified and studied in the course of the *SSICLOPS* project.

Following this paragraph, the partners involved in Task 2.2 provide a selected list of preceding contributions to the field of security research in the context of cloud computing. Afterwards, a comprehensive review of the state of the art is provided to form a body of knowledge.

3.1. Preceding contributions

Prior to the work on *SSICLOPS*, several aspects relevant to security-aware storage and processing have been researched at the Operating Systems and Middleware Group of the Hasso Plattner Institute. Among these aspects are technologies such as threshold cryptograph, trust-based access control, virtual machine introspection and searchable encryption. Since it is likely that our efforts on the *SSICLOPS* project will be able to build up on top of the insights gained in these preceding contributions, a brief overview is provided.

3.1.1. Threshold Cryptography

In a widely distributed environment, traditional authorization services represent a single-point of failure: If the service is unavailable, the encrypted data cannot be accessed by any party. In distributed setups, simple replication mechanisms can be considered a security threat, since

attackers can gain full control as soon as a single node has been compromised. In order to eliminate this weakness, the general approach presented by Neuhaus et al. (2012) [51] employs the concept of Fragmentation-Redundancy-Scattering [19]: Confidential information is broken up into insignificant pieces which can be distributed over several network nodes.

The contribution of Neuhaus et al. (2012) [51] is the design of a distributed authorization service. A system architecture has been presented that enables fine-grained access control on data stored in a distributed system. In order to maintain privacy in the presence of compromised parties, a threshold encryption scheme has been applied in order to limit the power of a single authorization service instance.

3.1.2. Trust-Based Access Control

The Operating System and Middleware Group operates a web platform called *InstantLab* [49, 50]. The purpose of the platform is to provide operating system experiments for student exercises in the undergraduate curriculum. Virtualization technology is used to provide pre-packaged experiments, which can be conducted through a terminal session in the browser. Thus far, massive open online-courses (MOOCs) have not been well suited for hands-on experiments, since assignments have been non-interactive. The main goal of *InstantLab* is to provide more interactive assignments and enable iterative test-and-improve software development cycles as well as observational assignments.

Providing a platform that enables a large audience to perform live software experiments creates several challenges regarding the security of such a platform. Malicious users might abuse resources for other means than the intended software experiments. In order to detect misuse of the provided resources, virtual machine introspection is applied. Furthermore, *InstantLab* [49, 50] demonstrates how automatic resource management is enabled by trust-based access control schemes. The purpose of trust-based access control is to restrict user access to resource intensive experiments. The approach implemented in *InstantLab* [49, 50] calculates a user's trust level based on his/her previous behavior.

3.1.3. Virtual Machine Introspection

In the age of cloud computing and virtualization, virtual machine introspection provides the means to inspect the state of virtual machines through a hypervisor without the risk of contaminating its state. Inspection capabilities are useful for a wide range of use case scenarios, ranging from forensics to more harmless cases such as making sure a tenant is not violating against the terms of use of the provider.

The work of Westphal et al. (2014) [68] contributes to the field of virtual machine introspection by providing a monitoring language called VMI-PL. Using this language, users can specify which information should be obtained from a virtual machine. Unlike competing approaches like

libVMI [39] and VProbes [66], VMI-PL does not limit users to hardware level metrics, but it also provides operating system level information such as running processes and other operating system events. Furthermore, the language can also be used to monitor data streams such as network traffic or user interaction.

3.1.4. Searchable Encryption

For many use cases, efficient and secure data sharing mechanisms are crucial, especially in distributed scenarios where multiple parties have to access the same data repositories from arbitrary locations. In such scenarios, the scalability of cloud computing makes resources simple to provision and to extend. However, when it comes to storing sensitive data in cloud-hosted data repositories, data confidentiality is still a major issue that discourages the use of cloud resources in sensitive scenarios. While traditional encryption can be used to protect the privacy of data, it also limits the set of operations that can be performed efficiently on encrypted data, such as search. Encryption schemes which allow the execution of arbitrary operations on encrypted data are still utopian. However, searchable encryption schemes exist that enable keyword-based search without the disclosure of keywords.

Neuhaus et al. (2015) [48] studied the practical applicability of searchable encryption for data archives in the cloud. For their evaluation, an implementation of Goh's searchable encryption scheme [25] was embedded into the document-based database MongoDB. With the encryption scheme in place, benchmarks revealed that the overhead for insertions is negligible compared to an unencrypted mode of operation. Search queries on the other hand come with a considerable overhead, since Goh's scheme [25] mandates a linear dependency between the complexity of search operations and the number of documents. However, the processing time of encrypted queries should be in acceptable orders of magnitude for interactive use cases where the increased security is mandatory.

3.2. State of the Art

In the context of cloud computing, the field of work related to security-aware storage and processing comprises a wide range of diverse directions. In the consequent part of this document, the state of the art is presented for a selection of differentiated topics. First, projects are highlighted which provide best practices for increasing security. Afterwards, new trends in virtualization strategies are outlined, followed by a brief introduction to novel hardware security mechanisms. Finally, the security aspects of providing coprocessor resources in virtual machines is illustrated.

3.2.1. New trends in virtualization strategies

Virtualization still remains as one of the main technological pillars of cloud computing. The main reason for this key role is that it enables high degrees of resource utilization and flexibility. Today, the most common approach for virtualization resorts to low-level hypervisors like *Xen* or *KVM* that employ hardware assisted virtualization in order to run regular guest operating systems in a para-virtualized or fully virtualized fashion. Recently however, new virtualization approaches have gained momentum. While containerization approaches move the scope of virtualization to higher levels of the application stack, unikernels are working at the same level of abstraction as regular operating systems but at the same time change the operating system drastically. A comparison of the different approaches is illustrated in Figure 2.

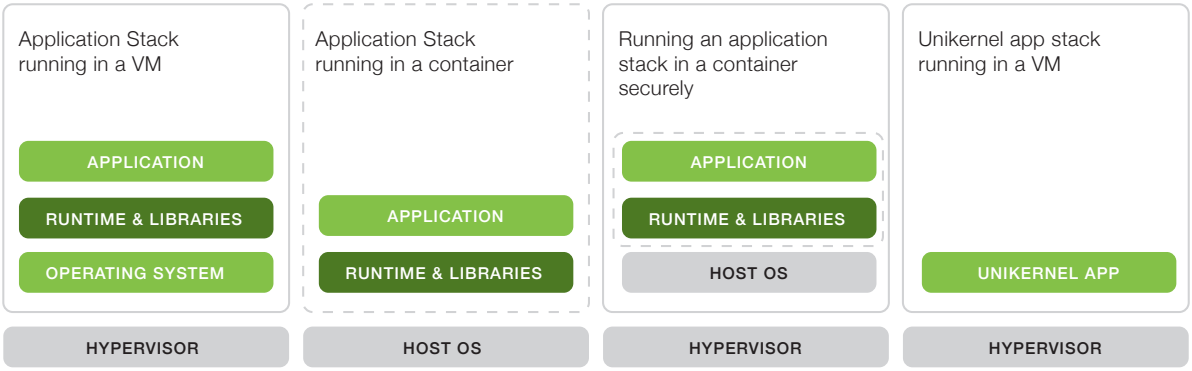

Figure 2: The regular guest virtual machine stack as well as both strategies for containerization come with a significant amount of overhead. Unikernels aim at reducing the footprint of virtualized applications. Source: [70]

3.2.1.1. Containers

In contrast to hypervisor-level virtualization approaches, where an entire operating system instance is virtualized, containers belong to the class of operating-system-level virtualization strategies that utilize multiple user-space instances in order to isolate tenants. The main goal of popular containerization implementations like Linux Containers [8] and Docker [12] is to reduce the memory footprint of hosted applications and to get rid of the overhead inherent to hypervisor-based virtualization. Recent studies demonstrate that the concept of containerization is able to outperform matured hypervisors in many use cases [20, 62, 69]. Regarding security aspects, most containerization approaches thus far rely on the operating system kernel to provide sufficient means of isolation between different containers. However, the LXD project [8] aims at providing hardware-based security features to containers in order to provide isolation levels on par with hypervisor-level based virtualization.

3.2.1.2. Unikernels

Unikernels are a new approach to hypervisor-level virtualization. The core concept of unikernels is based on the idea of deploying applications by merging application code and a minimal operating system kernel into a single immutable virtual machine image that is run on top of a standard hypervisor [41, 43, 60]. Since unikernels intentionally do not support the concept of process isolation, no time-consuming context switches have to be performed. The general idea behind unikernel systems is not entirely new, as it builds up on top of the concept of *library operating systems* such as exokernel [18] or Nemesis [59]. The main difference to library operating systems is that unikernels only run on hypervisors and do not support bare metal deployments, whereas library operating systems are targeting physical hardware. Due to the necessity to support physical hardware, library operating systems struggled with compatibility issues and proper resource isolation among applications. Unikernels are solving these problems by using a hypervisor in order to abstract from physical hardware and to provide strict resource isolation between applications [41]. Currently, the two most popular unikernel implementations are OSv [35] and Mirage OS [42].

According to Madhavapeddy et al. [41], unikernels are able to outperform regular operating systems in the following disciplines:

Boot time Unikernel systems are single purpose systems, meaning that they run only one application. Unnecessary overhead is stripped of by only linking libraries into a unikernel image which are required by the application. As a result, very fast boot times can be achieved. In their latest project *Jitsu: Just-In-Time Summoning of Unikernels* [10], Madhavapeddy et al. managed to achieve boot times in the order of 350ms on ARM CPUs and 30ms on x86 CPUs, which enables the possibility of dynamically bringing up virtual machines in response to network traffic.

Image size Since a unikernel system only contains the application and only the required functionality of the specialized operating system kernel, unikernel images are much smaller compared to traditional operating system images. The smaller binaries simplify management tasks like live-migration of running virtual machine instances.

Security By eliminating functionality which is not needed for the execution of an application inside a unikernel image, the attack surface of the system is reduced massively. Furthermore, the specialized operating system kernel of a unikernel image is usually written in the same high-level language as the application. The resulting absence of technology borders facilitates additional opportunities for code checking like static type checking and automated code checking. However, even if an attacker should manage to inject malicious code into a unikernel instance, it can only cause limited harm since no other application runs within the same image.

3.2.2. Hardware-based security mechanisms

3.2.2.1. Trusted Execution Technology (TXT)

The goal of Intel Trusted Execution Technology (TXT) [32] technology is that the user can verify if the operating system or its configuration was altered after the boot up. This requires a trusted platform module (TPM) which stores system indicators securely. The approach TXT is using is called dynamic root of trust measurement. For this methodology, the system can be brought into a clean state (SENTER instruction) after the firmware was loaded. In this approach as mentioned earlier only the operating system level software gets measured. These measurements can be compared with the original files / properties of the OS that have to be known beforehand.

3.2.2.2. Software Guard Extensions (SGX)

Sensitive tasks have to face an abundance of potential threats on both the software and the hardware level. On the hardware level, sensitive information such as encryption keys can be extracted from the systems main memory using DMA attacks or cold boot attacks. On the software level, the worst case has to be assumed and even the operating system has to be considered as a potential threat. While the concept of processes implements a high level of isolation between different applications, the elevated privileges of an operating system allow it to tamper with any process. These capabilities always pose a security threat, not just in the obvious case where the operating system might not be fully trusted. Even with a trusted operating system, there is always a certain risk that malicious code running in a separate process might gain elevated privileges. As soon as that happens, a malicious application can tamper with any process running on the system.

As a countermeasure to these threats, the Intel Software Guard Extensions (SGX) [46] introduced secure enclaves, which allow the safe execution of sensitive tasks even in untrustworthy environments. Enclaves are protected memory areas, which are encrypted and entirely isolated (see Figure 3). Even privileged code is not able to access the contents of an enclave. One process can even use multiple enclaves, which allows a high degree of flexibility. SGX does not require a trusted platform module (TPM), as the entire feature is implemented on the CPU. This level of integration reduces the list of trusted vendors to the CPU manufacturer and thus minimizes the number of potential attack vectors.

3.2.3. Virtualization of coprocessors resources

Coprocessors such as *Graphics Processing Units* (GPUs), *Field-Programmable Gate Arrays* (FPGAs) or Intel's *Many Integrated Core* (MIC) devices have become essential components in the *High Performance Computing* (HPC) field. Regarding the domain of cloud computing however, the utilization of coprocessors is not that well established. While there has been little demand for HPC-like applications on cloud resources in the past, the demand for running

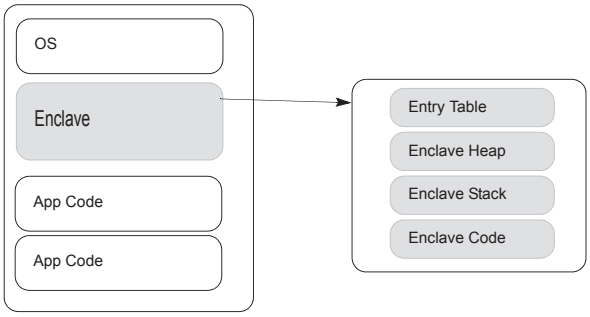

Figure 3: Secure enclaves provide an encrypted address space that is protected even from operating system access. Source: [31]

scientific applications on cloud computing infrastructure has increased [6]. Furthermore, moving compute-intensive applications to the cloud is becoming increasingly feasible [44] when it comes to CPU-based tasks. While several providers already offer cloud resources with integrated GPUs, their implementation is based on pass-through of native hardware. Assigning dedicated devices to each virtual machine results in high operational costs and decreased levels of flexibility. In such setups, virtual machines can neither be suspended nor migrated. Furthermore, the one-to-one mapping between pass-through devices and virtual machines prevent efficient utilization of coprocessors, which has a negative impact on cost effectiveness due to the high energy consumption of such hardware. Although projects exist that maximize resource utilization by providing unused GPU resources to other compute nodes [45] or that save energy by shutting down inactive compute nodes [38], a large gap between the capabilities of GPU and CPU virtualization still exists.

In the ensuing paragraphs, the state of the art of coprocessor virtualization is evaluated based on several characteristics. Most work deals with GPU compute devices, however the general techniques are applicable to other coprocessor classes as well. For desktop-based GPU virtualization, Dowty and Sugerman [13] define four characteristics that are to be considered: performance, fidelity, multiplexing and interposition. Regarding cloud-based virtualization, the aforementioned enumeration is missing isolation as an important characteristic. In order to establish coprocessors in cloud computing, one of the most crucial characteristics is that multiple tenants have to be properly isolated. Since the focus is set on coprocessors and thus compute-based capabilities instead of interactive graphics, fidelity can mostly be ignored for our use case. Last but not least, performance should not suffer severely from the virtualization overhead. However, without isolation, multiplexing and interposition capabilities, performance is worthless in the cloud computing use case.

3.2.3.1. Isolation

Thus far, isolation is only addressed by approaches that make use of mediated pass-through strategies like Intel GVT-g (formerly called gVirt) [64] and NVIDIA GRID [52]. While the latter is a commercial closed-source implementation, the implementation details of GVT-g are publicly available as an open source project. In Intel's approach, each virtual machine runs the native graphics driver. In contrast to regular pass-through, mediated pass-through uses a trap-and-emulate mechanism is used to isolate virtual machine instances from each other. The main drawback is that the implementation of the mediated pass-through strategy has to be tailored to the specifications of each supported GPU, which again requires detailed knowledge about the GPU design. Overall, this approach is only feasible for the manufacturers of GPUs themselves.

With rCUDA [14–16], vCUDA [61], gVirtuS [23], GViM [26], VirtualCL [4] and VOCL [71], many approaches exist which are based on call forwarding. Originating from the field of High Performance Computing, the call forwarding approach uses a driver stub in the guest operating system which redirects the calls to a native device driver in the privileged domain. Since isolation is barely an issue in the HPC domain, none of the existing approaches implement isolation mechanisms.

3.2.3.2. Multiplexing

Sharing a single GPU among multiple virtual machines is possible for all aforementioned implementation strategies. In the faction of mediated pass-through implementations, both Intel GVT-g and NVIDIA GRID support multiplexing in order to serve multiple virtual machines with a single GPU. As for isolation, a trap-and-emulate mechanism in the hypervisor coordinates devices accesses from multiple virtual machines. On the side of call forwarding approaches, the implementation of multiplexing capabilities with low overhead is very a tough challenge. In the privileged domain, additional logic has to be implemented that schedules requests from different guests. So far, only vCUDA [61] provides such multiplexing mechanisms.

3.2.3.3. Interposition

While mediated pass-through approaches excel call forwarding strategies in both isolation and multiplexing, interposition is hard to achieve for mediated pass-through. Although an implementation is possible in theory [73], it is not feasible in practice as it is susceptible to the slightest variations on the hardware level. With vCUDA [61] and VOCL [71] on the other hand, multiple projects based on call forwarding exist that successfully implement interposition capabilities. Again, a piece of middleware is required in the hypervisor which carefully tracks the state of each virtual GPU instance. With such capabilities at hands, virtual machines can be suspended and even live-migrated to other virtual machine hosts.

3.2.4. Security-oriented best practices collections

Over the last years, several best practice collections and frameworks dealing with improving the security of information technology were established and maintained by companies and public authorities alike.

3.2.4.1. Critical Security Controls

The term "Security Fog of More" was established by Tony Sager, a chief technologist of the *Council on CyberSecurity*. He noticed that security professionals are confronted with a plethora of security products and services. These choices are influenced by compliance, regulations, frameworks and audits e.g. the "Security Fog of More". As a consequence, one of the main challenges today is making an educated choice. Sager wants to help security professionals by providing a framework for security choices called *Critical Security Controls* [10] (see Figure 4) that spans 20 different areas of IT security containing suggestions for each of these areas with a focus on scalability of the solutions.

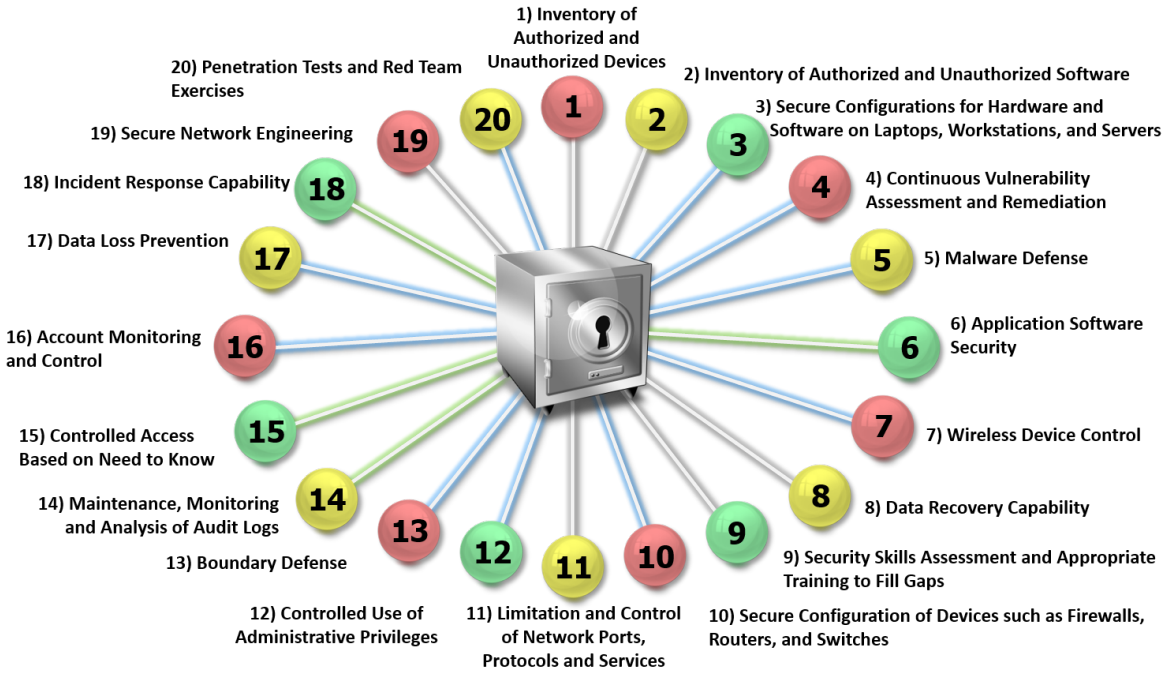

Figure 4: The Critical Security Controls framework categorizes security threats in 20 classes. Source: [30]

3.2.4.2. Open Web Application Security Project

The Open Web Application Security Project (OWASP) [54] is a non-profit organization founded 2004 with the goal of improving software security. The OWASP houses a wide range of security

related projects centered around all aspects of software development driven by a large community of volunteers. We will provide a brief overview over a selection of the most popular OWASP projects:

OWASP Developer Guide The OWASP Developer Guide [55] was the first project pursued by OWASP. In its latest revision, the guide describes general concepts about developing secure software without a focus on specific technologies. The guide covers topics such as Architecture, Design, Build Process and Configuration of secure software and is targeting developers as its target audience. The instructions can be used as additional guidelines for penetration testers as well.

OWASP Testing Guide The OWASP Testing Guide [56] is a best practice collection for penetration testing of web applications and services. The guide covers the software development process as well as testing approaches for different parts of web applications (e.g. Authentication, Encryption or Input Validation).

OWASP Top 10 The OWASP Top 10 [57] is a list maintained by security experts which contains the 10 most prevalent security flaws in web applications. The goal of this list to establish a security awareness in IT companies to prevent the occurrence of the most common vulnerabilities in their applications.

3.3. Discussion

The state of the art presented in the previous section has demonstrated that a vast variety of approaches exist that are accommodated by the topic *Security-Aware Storage and Processing*. While soft approaches such as best practice collections are beneficial for everyday use, they are of limited use for technically oriented research prototypes. On the other end of the scale are hardware security features such as TXT and SGX.

The Software Guard Extensions is an interesting new feature that can be used to evaluate problems that require the execution of crucial code in untrustworthy requirements. However, it should be noted that until the day of writing, no commercially available processor implements the SGX feature. Moreover, it is even unclear when such a processor can be expected to become available. Under the bottom line, it seems like SGX provides various research opportunities, however the focus for near future projects should be shifted to different topics.

With virtualization being a key technology in cloud computing, it is important to keep an eye on new virtualization concepts. With the advent of containerization, a new approach to virtualization has surfaced that tries to minimize the performance overhead caused by an additional level of context switches. While containers have already achieved a certain prevalence rate, unikernels are

a recent re-discovery of an old concept. Unikernels should be considered as a direct competition to containers, since they also address mitigation of virtualization overhead while maintaining a thorough level of isolation. Even though there is a certain risk that unikernels might be a fashionable trend, eventual benefits over traditional virtual machines and containerization approaches should be evaluated. With boot times of tens of milliseconds, the use of unikernels might enable new degrees of dynamic resource utilization and improved power management.

In the subject area of virtualization, server-based virtualization of coprocessor resources, most importantly *Graphics Processing Units* (GPUs), is another aspect that has been neglected in the past. High operational costs are caused by poorly utilized devices. Even though some approaches exist that allow resource multiplexing, the near absence of proper isolation has been a deal-breaker for the cloud computing scenario thus far.

4. Conclusion

In this deliverable, we presented the state of the art of privacy policy languages with respect to the scenario of cloud computing. We showed that the cloud demands for a privacy policy language to enable users to specify their privacy preferences to protect their personal data. Furthermore, we derived requirements that a privacy policy language in the cloud setting must fulfill, i.e., we showed the requirement of (i) a small storage footprint, (ii) readability, (iii) comprehensibility of the evaluation process, (iv) conflict detection or conflict handling, (v) performance, (vi) expressiveness, (vii) extensibility, (viii) policy enforcement, (ix) support of developers, and (x) matching of user-policies and service-policies. Following, we presented existing policy languages with respect to these requirements, i.e., we described the existing mechanisms employed to fulfill the derived requirements. Afterwards, we discussed the importance of the different derived requirements to obtain a guideline in case of conflicting requirements. We argued for the importance of (i), (ii), (iv), (v), (vi), (vii) and (x), derived that (iii), (x), (ix) are beneficial but not mandatory, and showed that (iv) is tackled best with the help of conflict detection at time of policy specification. Furthermore, we outlined mechanisms that are valuable to achieve listed requirements but are not employed by the state of the art. Finally, we showed that existing languages, while occasionally introducing valuable mechanisms, in their entirety do not fit for the scenario of cloud computing.

Regarding the aspect of security-aware storage and processing, a plethora of leverage points exists to address the subject. In order to set a focus to a limited subset of approaches, we evaluated a selected list of directions that deal with security. We provided an overview over current activities in (i) unikernels and containers as alternatives to regular virtual machines, (ii) hardware security extensions that allow secure execution in untrustworthy environments, (iii) the lack of proper virtualization capabilities of coprocessors resources and (iv) security-oriented best practices collections.

5. Outlook

Our discussion of the state of the art of policy languages showed that no existing privacy policy language fully supports the scenario of cloud computing. However, some mechanisms are valuable and may be combined with mechanisms we outlined during our discussion of the requirements. Nevertheless, the scenario of cloud computing demands for a privacy policy language specifically tailored for its needs. To this end, this privacy policy language must provide a small storage footprint to limit overhead of cloud services, and an easy to read and understand abstraction for the definition and understanding of policies by non-experts. It must support detection of conflicts at time of specification to enable explicit handling by the user, and provide a performance that is suitable for application at less powerful middleboxes. Thereby, the language must be able to express obligations, triggers, be agnostic to the environmental context and provide the ability to extend its functionality to cope with emerging cloud services. Finally, the matching of user-specified policies and policies of cloud services should be integrated into the evaluation process to obtain synergies regarding performance. Overall, the presented state of the art and the discussion provide a valuable basis for the development of a privacy policy language that is tailored to enable users to specify their privacy demands in the scenario of cloud computing.

Recalling the directions of security research discussed in Sections 3.2 and 3.3, many approaches exist for providing increased levels of security in the use case of cloud computing. While best practices collections may be beneficial for everyday use, they are of limited use for technically oriented research interests. It seems like technological improvements like the Software Guarded Extensions (SGX) are an interesting target for further research efforts. However, the uncertain date of availability of the technology enforces a postponed examination of the topic. Regarding new virtualization approaches, there is a certain risk that unikernels are a fashionable trend that might disappear rather sooner than later. However, the crucial role of virtualization in cloud computing suggests that unikernels and containers should be evaluated more thoroughly. From a functional perspective, these new virtualization approaches do have the potential to improve both, security aspects and performance. Regarding the non-function side of the topic, unikernels might enable us to improve both dynamic resource utilization and power management strategies. Last but not least, employing coprocessor resources in cloud computing is a topic that requires extensive research efforts. In order to move from dedicated devices to truly shared resources, security is a major concern that has not been solved yet. Lightweight isolation mechanisms have to be researched that provide tight levels of isolation while inducing bearable levels of overhead compared to native hardware.

A. Appendix: Relation to other EU Projects and Standards

A.1. Benefits over alternative Policy Languages

In this deliverable, we extensively analyzed existing policy languages with respect to their application in (federated) cloud computing. We now extend our analysis with more detailed information regarding achievements of finished and concurrent EU projects as well as established standards. Furthermore, we compare the scope and results of these EU projects with the scope, goals, and achievements that we already realized with our policy language work in the SSICLOPS project.

A.1.1. XML-based Standard

The Topology and Orchestration Specification for Cloud Applications (TOSCA) [53] defines an XML-based language to describe service components and their relations as well as their creation, modification, and termination. It also supports the specification of requirements that components impose on the cloud environment which hosts them. Thus, the described language allows for specification of expectations regarding services that a user or company deploys in a cloud environment. In SSICLOPS, we instead focus on policies for data items that are stored and processed in the cloud.

TOSCA also supports the definition of *invariant properties* in *policy templates* which are set by an actual *policy*. This concept is similar to the domain knowledge approach that we pursue with SSICLOPS' Compact Privacy Policy Language (CPPL) [28]. This domain knowledge specifies the properties a user can use to express policies. Thereby, similar to invariant properties in TOSCA, it also puts some restrictions on values that a variable can take. However, TOSCA merely uses this approach to create templates for policies. With CPPL, we also feature such a template functionality as the domain knowledge defines the usable variables and the values that they can take – which may be a restricted set such as specific numbers up to freely specifiable strings – but, more importantly, we also leverage the domain knowledge to significantly reduce the storage footprint with our domain knowledge based compression. Notably, the XML-based TOSCA specification suffers from a comparable large storage footprint due to the XML encoding which CPPL significantly outperforms [28]. Thus, CPPL proves much more valuable for the high-frequent processing of the massive amount of data processed in (federated) clouds.

A.1.2. XACML-based Policy Languages

The PrimeLife[1] project of the EU's 7th Framework Programme funded the research of Ali et al. [2]. As our discussion of this work in Section 2.3.2 shows, the proposed language does not provide the necessary features for application in the scenario of federated cloud computing, especially as it merely focuses on expressiveness while neglecting important properties such as storage footprint, readability, and performance and does not provide a mechanism for matching of user expectations and provider policies.

The PrimeLife project also developed the *PrimeLife Policy Language* (PPL) [7]. This policy language extends XACML [24] (cf. Section 2.2.1) with time-based and action-based *triggers*. Furthermore, PPL defines a mechanism for matching of user expectations and cloud provider policies. However, as PPL is still based on XACML, it inherits corresponding issues regarding storage footprint and readability. Moreover, PPL misses an evaluation of its runtime performance which is a key requirement for a policy language used for data annotations in the scenario of (federated) cloud computing. With our policy language work in the SSICLOPS project, we target to overcome especially the storage footprint limitations of XACML and PPL, i.e., we develop a policy language that does not only support the expressiveness needed for federated cloud computing but also allows for efficient transmission and fast processing of data annotated with policies. To this end, we introduced CPPL [17, 28] which features a domain-based compression mechanism that still allows for high-performance evaluation of policies. Furthermore, we explicitly showed that CPPL is able to express PPL policies and significantly improves the storage footprint compared to the representation in PPL [28].

The *Accountability Policy Language* (A-PPL) [3] is an extension of PPL developed by the A4Cloud[2] project under EU's 7th Framework Programme. It specifies extensions for PPL's syntax to enable *access control rules*, *data retention*, *reporting and notification*, *controlling data location*, *auditability*, and *logging*. However, A-PPL still bases upon XACML and thus suffers from the same problems regarding storage footprint and readability when used in the context of (federated) cloud computing. Furthermore, it also does not account for the performance requirements of this domain. SSICLOPS' policy language CPPL can easily express A-PPL policies and thereby significantly improves the storage footprint and achieves suitable performance [28].

A.1.3. Limited Functionality

The PaaSword project[3] funded by EU's Horizon 2020 Programme develops security annotations to protect data in the cloud with access control, cryptographic protection, and physical distribution. To this end, PaaSword utilizes context-aware access control mechanisms and searchable encryption schemes, which can process search queries on encrypted data [47]. Nevertheless, the limitation to perform only searches over encrypted data puts significant boundaries on the tasks that the cloud

[1] http://primelife.ercim.eu/
[2] http://www.a4cloud.eu/
[3] https://www.paasword.eu/

is able to perform. For other operations, the cloud still needs access to the decrypted data. With CPPL, we take a broader approach and specify data annotations for arbitrary user expectations, i.e., CPPL can restrict the processing to searches on encrypted data as the PaaSword approach, but also enables users to specify their expectations regarding processing tasks that require the cloud to get access to decrypted user data.

We note that the PaaSword project could use homomorphic encryption approaches that enable the cloud to perform processing beyond search queries on encrypted data. However, solutions that allow for general processing on encrypted data still suffer from significant performance issues that make these mechanisms unfeasible for the large scale application in cloud computing today [22].

Furthermore, PaaSword does not support only limited functionality but also bases its solution on XACML which does not fulfill the requirements regarding storage footprint and performance in federated cloud computing scenarios (cf. Section 2.3.2). Thus, even when neglecting the general limitations of PaaSwords security annotations compared to CPPL, the storage footprint and runtime performance of this approach could significantly benefit from the mechanisms introduced by CPPL [28].

A.1.4. Summary

Summing up our analysis of policy languages developed in related EU projects and standardization efforts, these policy languages often focus on the realization of functionality. However, they neglect requirements such as storage footprint and runtime performance that become relevant when analyzing policy languages from a networking perspective. This especially holds for the usage of policy languages used for data in highly interconnected cloud instances in the scenario of federated cloud computing. Other approaches specifically tailor policy languages for certain scenarios which yields a limited scope of application. In SSICLOPS, we address the neglected network-related requirements as well as the flexible adaption to arbitrary use cases with the domain knowledge mechanism and efficient policy encoding, both presented as part of our policy language CPPL.

A.2. Relation to Concurrent Work

Apart from alternative policy languages, there exists concurrent work on enabling user expectation aware processing of data in the cloud. However, as many of the policy languages above, also these works neglect the requirement for an efficient realization of such a support.

A.2.1. Negotiation of SLAs

SLA-Ready[4] is a Horizon 2020 project targeting to enhance the understanding of Service Level Agreements (SLAs) by small and medium sized enterprises (SMEs). This work bases on the assumption that SMEs, while being one of the main beneficiaries of cloud computing, refuse to use cloud services due to the missing understanding of underlying SLAs and corresponding security and privacy concerns. To improve this situation, SLA-Ready targets to assist SMEs to understand SLAs with the help of a better standardization such that SMEs can take an informed decision. To this end, SLA-Ready establishes best practices and develops a *common reference model* with the target to establish a common vocabulary and enabling comparison of SLAs [40, 65]. This common reference model will develop into an *SLA Marketplace* which provides SMEs access to the information and tools developed by SLA-Ready.

On the technical side of SLA negotiation, the SLALOM project[5] targets the standardization of SLA negotiation for the usage of cloud services. To this end, SLALOM defines a negotiation process that is compliant to the current version of ISO 19086-2. In contrast to most other approaches, which base upon XACML, SLALOM's SLA negotiation bases upon JSON. While JSON features a smaller storage footprint compared to XML as used by XACML, it, however, still suffers from a large storage footprint compared to that achieved by SSICLOPS' CPPL which performs encoding optimization on the bit-level. At the same time, the domain knowledge approach of CPPL provides sufficient expressiveness for real world policies [28]. Further comparing it to SSICLOPS, SLALOM focuses on the enabling of SLA negotiation between entities in the cloud environment. The policy work in SSICLOPS instead especially focuses on the efficient technical realization of policy communication between the different entities in (federated) clouds. This efficiency is paramount to enable feasible use of policies that is challenged by a huge amount of data to be processed [28].

A.2.2. Hiding Policies from the Cloud: Proxy-based Approaches

The CLARUS project[6] as well as the SUPERCLOUD project[7], both funded by EU's Horizon 2020 Programme, develop proxies that secure user data before it is stored in the cloud. While other approaches require support by the cloud provider to apply user policies, a proxy that handle data and prepares it according to the users expectations before it is sent to the cloud does not require the cloud provider to understand or process user policies.

However, proxy approaches face are significantly limited in their functionality. Hiding user expectations from cloud providers limits the security methods that are applicable to protect stored data to encryption and distribution to different cloud instances. More specifically, proxy-based approaches can only encrypt data at the client or proxy before sending it to the cloud and

[4] http://www.sla-ready.eu/
[5] http://slalom-project.eu/
[6] http://clarussecure.eu/
[7] https://supercloud-project.eu/

the proxy can distribute data to clouds that are under control of different entities. As fully homomorphic encryption is still not feasible today [22], processing of data by cloud instances is thus limited to specialized methods, e.g., searchable encryption, specialized homomorphic encryption algorithms, and secure two-party or secure multi-party computation. Thus, the proxy approach considerably limits the processing capabilities of cloud instances on data stored by the cloud. In contrast, approaches that make the cloud aware of the user policy do not suffer from this limitation. Nevertheless, hiding user expectations from cloud providers may be required in specific use cases where the user policy itself must be hidden from the cloud provider for privacy reasons. Notably, while proxy-based approaches allow for the usage of multiple cloud providers, this is likewise enabled by federated clouds that we focus in the context of SSICLOPS. Finally, realizing policies with a proxy is still limited by the capabilities offered by cloud providers and therefore not fully independent of actual support by the cloud.

Still, proxy-based approaches could benefit from CPPL. In cases where a proxy should handle policies, e.g., to hide policies from cloud providers, clients could use CPPL to efficiently communicate these policies to the proxy. This would be especially useful if the proxy is offered as a service by a trusted third party, i.e., the proxy must handle a lot of data and therefore significantly profits from the increased efficiency that comes with the approaches introduced with CPPL. Furthermore, the usage of CPPL to communicate requirements to the proxy would result in a uniform mechanism that enables users to specify policies that should be kept hidden from cloud providers as well as policies that can be transmitted to and handled by the cloud.

A.3. Complementary Work

Realizing the support and usage of policies in cloud environments requires private and enterprise users of cloud systems to be aware of the security and privacy issues and must enable them to efficiently negotiate SLAs and policies with cloud providers. However, policy handling does not end at the border of cloud environments, but also requires adapting mechanisms within the cloud.

A.3.1. Enforcing Cloud Provider Adherence to Policies

Our work regarding policy support focuses on its efficient realization for (federated) cloud computing. More specifically, our policy language CPPL does not provide mechanisms to enforce the handling of data by cloud providers according to negotiated policies. Nevertheless, users often do not trust cloud providers and therefore require that they attest their benign behavior, i.e., their adherence to specified policies.

To realize control on the actions of cloud providers, the CUMULUS project[8] (funded by EU's 7th Framework Programme) developed mechanisms for certification of cloud instances. This

[8] http://www.cumulus-project.eu/

certification can range from testing the delivery of negotiated performance up to certification of the full system based on trusted computing platforms. Similarly, the project SPECS[9] (funded by EU's 7th Framework Programme) developed a framework that monitors cloud instances to detect deviation from negotiated SLAs. Such mechanisms can be combined with policy negotiation approaches such as CPPL to enforce the handling of data according to the previously negotiated user policies or SLAs.

The SERECA project[10] funded by EU's Horizon 2020 Programme is another project focusing on establishing trust in the computation at devices outside of the users control sphere. It investigates secure enclaves as technical means for enabling applications with strict confidentiality and integrity policies in potentially insecure execution environments. These secure enclaves are facilitated by technologies such as ARM *TrustZone* or Intel *Software Guard Extensions* (SGX) and provide applications with isolated address spaces that are protected even against access from privileged code such as operating systems or hypervisors. Similarly, the SecureCloud project [11] (funded by the EU's Horizon 2020 Programme) aims at leveraging the Intel SGX technology to facilitate secure enclaves within container-based units of deployment. With these hardware-supported security mechanisms at hands, it might be feasible to implement hardened policy enforcement points.

[9] http://www.specs-project.eu/
[10] http://www.serecaproject.eu
[11] https://www.securecloudproject.eu

Bibliography

[1] M. Ali, L. Bussard, and U. Pinsdorf. "Obligation Language and Framework to Enable Privacy-Aware SOA". English. *Data Privacy Management and Autonomous Spontaneous Security*. Ed. by J. Garcia-Alfaro, G. Navarro-Arribas, N. Cuppens-Boulahia, and Y. Roudier. Vol. 5939. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 18–32. DOI: 10.1007/978-3-642-11207-2_3.

[2] M. Ali, L. Bussard, and U. Pinsdorf. "Obligation language and framework to enable privacy-aware SOA". *DPM*. 2009.

[3] M. Azraoui, K. Elkhiyaoui, M. Önen, K. Bernsmed, A. S. Oliveira, and J. Sendor. "A-PPL: An accountability policy language". *DPM*. 2014.

[4] A. Barak and A. Shiloh. "The VirtualCL (VCL) Cluster Platform". ().

[5] M. Y. Becker, A. Malkis, and L. Bussard. "A Practical Generic Privacy Language". English. *Information Systems Security*. Ed. by S. Jha and A. Mathuria. Vol. 6503. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 125–139. DOI: 10.1007/978-3-642-17714-9_10.

[6] S. Benedict. "Performance issues and performance analysis tools for HPC cloud applications: a survey". *Computing* 95.2 (2013), pp. 89–108.

[7] L. Bussard, G. Neven, and F. S. Preiss. "Downstream usage control". *POLICY*. 2010.

[8] Canonical Ltd. *Linux Containers Project*. https://linuxcontainers.org. Accessed: 2015-07-15.

[9] L. Chen, C. J. Mitchell, and A. Martin, eds. *Trusted Computing*. en. Vol. 5471. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009. URL: http://link.springer.com/10.1007/978-3-642-00587-9 (visited on 07/30/2015).

[10] Council on CyberSecurity. *The Critical Security Controls for Effective Cyber Defense Version 5.0*. Tech. rep. https://www.sans.org/, Feb. 2014.

[11] A. Dehghantanha, N. Udzir, and R. Mahmod. "Towards a Pervasive Formal Privacy Language". *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*. Apr. 2010, pp. 1085–1091. DOI: 10.1109/WAINA.2010.26.

[12] Docker, Inc. *Docker Project*. https://www.docker.com. Accessed: 2015-07-15.

[13] M. Dowty and J. Sugerman. "GPU virtualization on VMware's hosted I/O architecture". *ACM SIGOPS Operating Systems Review* 43.3 (2009), pp. 73–82.

[14] J. Duato, F. D. Igual, R. Mayo, A. J. Peña, E. S. Quintana-Ortí, and F. Silla. "An Efficient Implementation of GPU Virtualization in High Performance Clusters". *Euro-Par 2009 - Parallel Processing Workshops*. Ed. by H.-X. Lin, M. Alexander, M. Forsell, A. Knüpfer, R. Prodan, L. Sousa, and A. Streit. Vol. 6043. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 385–394. DOI: 10.1007/978-3-642-14122-5.

[15] J. Duato, A. J. Pena, F. Silla, J. C. Fernandez, R. Mayo, and E. S. Quintana-Ortí. "Enabling CUDA acceleration within virtual machines using rCUDA". English. *2011 18th International Conference on High Performance Computing*. IEEE, Dec. 2011, pp. 1–10. DOI: 10.1109/HiPC.2011.6152718.

[16] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Ortí. "rCUDA: Reducing the number of GPU-based accelerators in high performance clusters". English. *2010 International Conference on High Performance Computing & Simulation*. IEEE, June 2010, pp. 224–231. DOI: 10.1109/HPCS.2010.5547126.

[17] F. Eberhardt, J. Hiller, S. Klauck, M. Plauth, A. Polze, and K. Wehrle. *SSICLOPS D2.2: Design of Inter-Cloud Security Policies, Architecture, and Annotations for Data Storage*. Tech. rep. Jan. 2016.

[18] D. R. Engler, M. F. Kaashoek, et al. *Exokernel: An operating system architecture for application-level resource management*. Vol. 29. 5. ACM, 1995.

[19] J.-C. Fabre, Y. Deswarte, and B. Randell. *Designing secure and reliable applications using fragmentation-redundancy-scattering: an object-oriented approach*. Springer, 1994.

[20] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. *An updated performance comparison of virtual machines and linux containers*. Tech. rep. 2014, p. 32.

[21] O. Garcia-Morchon and K. Wehrle. "Modular Context-aware Access Control for Medical Sensor Networks". *Proc. 15th ACM Symposium on Access Control Models and Technologies*. SACMAT '10. ACM, 2010, pp. 129–138. DOI: 10.1145/1809842.1809864.

[22] C. Gentry, S. Halevi, and N. P. Smart. "Homomorphic Evaluation of the AES Circuit". *Advances in Cryptology – CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Ed. by R. Safavi-Naini and R. Canetti. Springer Berlin Heidelberg, 2012, pp. 850–867. DOI: 10.1007/978-3-642-32009-5_49.

[23] G. Giunta, R. Montella, G. Agrillo, and G. Coviello. "A GPGPU transparent virtualization component for high performance computing clouds". *Euro-Par 2010-Parallel Processing*. Springer, 2010, pp. 379–391.

[24] S. Godik, A. Anderson, B. Parducci, P. Humenn, and S. Vajjhala. *OASIS eXtensible access control 2 markup language (XACML) 3*. Tech. rep. Tech. rep., OASIS, 2002.

[25] E.-J. Goh et al. "Secure Indexes." *IACR Cryptology ePrint Archive* 2003 (2003), p. 216.

[26] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan. "GViM: GPU-accelerated Virtual Machines Vishakha". *Proc. 3rd ACM Workshop on System-level Virtualization for High Performance Computing - HPCVirt '09*. ACM Press, Mar. 2009, pp. 17–24. DOI: 10.1145/1519138.1519141.

[27] M. Henze, R. Hummen, and K. Wehrle. "The Cloud Needs Cross-Layer Data Handling Annotations". *Security and Privacy Workshops (SPW), 2013 IEEE*. May 2013, pp. 18–22.

[28] M. Henze, J. Hiller, S. Schmerling, J. H. Ziegeldorf, and K. Wehrle. "CPPL: Compact Privacy Policy Language". *Proc. 2016 ACM on Workshop on Privacy in the Electronic Society*. WPES '16. ACM, 2016, pp. 99–110. DOI: 10.1145/2994620.2994627.

[29] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. "A Policy Language for Distributed Usage Control". English. *Computer Security – ESORICS 2007*. Ed. by J. Biskup and J. López. Vol. 4734. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 531–546. DOI: 10.1007/978-3-540-74835-9_35.

[30] F. T. Insider. *Continuous Diagnostics and Mitigation Addresses "Foundational" Issues Identified by SANS. http://www.federaltechnologyinsider.com/cdm-addresses-foundational-issues-identified-sans/*. May 2014. (Visited on 06/01/2015).

[31] Intel Corporation. *Intel© Software Guard Extensions Programming Reference*. Tech. rep. Oct. 2014.

[32] Intel Corporation. *Intel Trusted Execution Technology White Paper*. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf. Online, Accessed 31.07.2015.

[33] H. Jäger, A. Monitzer, R. Rieken, E. Ernst, and K. Nguyen. "Sealed Cloud - A Novel Approach to Safeguard against Insider Attacks". English. *Trusted Cloud Computing*. Ed. by H. Krcmar, R. Reussner, and B. Rumpe. Springer International Publishing, 2014, pp. 15–34. DOI: 10.1007/978-3-319-12718-7_2.

[34] L. Kagal, T. Finin, and A. Joshi. "A policy language for a pervasive computing environment". *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*. June 2003, pp. 63–74. DOI: 10.1109/POLICY.2003.1206958.

[35] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti, and V. Zolotarov. "OSv—optimizing the operating system for virtual machines". *2014 usenix annual technical conference (usenix atc 14)*. Vol. 1. USENIX Association. 2014, pp. 61–72.

[36] H. Krcmar, R. Reussner, and B. Rumpe, eds. *Trusted Cloud Computing*. en. Springer International Publishing, 2014. URL: http://link.springer.com/10.1007/978-3-319-12718-7 (visited on 07/30/2015).

[37] P. Kumaraguru, L. Cranor, J. Lobo, and S. Calo. "A survey of privacy policy languages". *SOUPS'07: Proceedings of the 3rd Symposium on Usable Privacy and Security*. 2007.

[38] P. Lama, Y. Li, A. M. Aji, P. Balaji, J. Dinan, S. Xiao, Y. Zhang, W.-c. Feng, R. Thakur, and X. Zhou. "pVOCL: Power-Aware Dynamic Placement and Migration in Virtualized GPU Environments". English. *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, July 2013, pp. 145–154. DOI: 10.1109/ICDCS.2013.51.

[39] LibVMI Project. *LibVMI*. http://libvmi.com. Accessed: 2015-07-17.

[40] J. Luna, D. Catteddu, CSA, S. Parker, and Trust-IT. *SLA-Ready Hub and Social Marketplace - first iteration*. Tech. rep. Apr. 2016.

[41] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. "Unikernels: Library operating systems for the cloud". *ACM SIGPLAN Notices*. Vol. 48. 4. ACM. 2013, pp. 461–472.

[42] A. Madhavapeddy, R. Mortier, R. Sohan, T. Gazagnaire, S. Hand, T. Deegan, D. McAuley, and J. Crowcroft. "Turning down the LAMP: software specialisation for the cloud". *Proc. 2nd USENIX conference on Hot topics in cloud computing, HotCloud*. Vol. 10. 2010, pp. 11–11.

[43] A. Madhavapeddy and D. J. Scott. "Unikernels: Rise of the virtual library operating system". *Queue* 11.11 (2013), p. 30.

[44] K. Mantripragada, A. Binotto, L. Tizzei, and M. Netto. "A Feasibility Study of Using HPC Cloud Environment for Seismic Exploration". *77th EAGE Conference and Exhibition 2015*. 2015.

[45] P. Markthub, A. Nomura, and S. Matsuoka. "Using rCUDA to Reduce GPU Resource-assignment Fragmentation caused by Job Scheduler". *15th International Conference on Parallel and Distributed Computing, Applications and Technologies*. 2014. DOI: 10.1109/PDCAT.2014.26.

[46] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. "Innovative Instructions and Software Model for Isolated Execution". *Proc. 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. ACM, 2013, 10:1–10:1. DOI: 10.1145/2487726.2488368.

[47] A. Michalas, S. Veloudis, Y. Verginadis, M. Spyros, R. Dowsley, G. Ledakis, S. Mantzouratos, J. Vuong, G. Schiefer, P. Gouvas, S. Nechifor, and E. Papatheocharous. *PaaSword Technical Requirements Deliverable 1.1*. Tech. rep. June 2016.

[48] C. Neuhaus, F. Feinbube, D. Janusz, and A. Polze. "Secure Keyword Search over Data Archives in the Cloud: Performance and Security Aspects of Searchable Encryption". *5th International Conference on Cloud Computing and Services Science,* ACM. 2015.

[49] C. Neuhaus, F. Feinbube, and A. Polze. "A Platform for Interactive Software Experiments in Massive Open Online Courses". *Journal of Integrated Design and Process Science* 18.1 (2014), pp. 69–87.

[50] C. Neuhaus, F. Feinbube, A. Polze, and A. Retik. "Scaling Software Experiments to the Thousands". *CSEDU 2014 - Proc. 6th International Conference on Computer Supported Education, Volume 1, Barcelona, Spain, 1-3 April, 2014*. 2014, pp. 594–601.

[51] C. Neuhaus, M. von Löwis, and A. Polze. "A Dependable and Secure Authorisation Service in the Cloud". *CLOSER*. 2012, pp. 568–573.

[52] NVIDIA Corp. *Shared Virtual GPU (vGPU) Technology*. URL: http://www.nvidia.com/object/virtual-gpus.html (visited on 01/11/2015).

[53] OASIS Standard. *Topology and Orchestration Specification for Cloud Applications Version 1.0*. 2013.

[54] OWASP Foundation. *Open Web Application Security Project (OWASP)*. http://www.owasp.org/. Accessed: 2015-07-02.

[55] OWASP Foundation. *OWASP Developer Guide*. https://www.owasp.org/index.php/Category:OWASP_Guide_Project. Accessed: 2015-07-02.

[56] OWASP Foundation. *OWASP Testing Guide*. https://www.owasp.org/index.php/Category:OWASP_Testing_Project. Accessed: 2015-07-02.

[57] OWASP Foundation. *OWASP Top 10*. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. Accessed: 2015-07-02.

[58] J. Poroor and B. Jayaraman. "C2L:A Formal Policy Language for Secure Cloud Configurations". *Procedia Computer Science* 10 (2012). {ANT} 2012 and MobiWIS 2012, pp. 499–506. DOI: http://dx.doi.org/10.1016/j.procs.2012.06.064.

[59] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt. "Rethinking the library OS from the top down". *ACM SIGPLAN Notices* 46.3 (2011), pp. 291–304.

[60] D. Schatzberg, J. Cadden, O. Krieger, and J. Appavoo. "A way forward: enabling operating system innovation in the cloud". *Proc. 6th USENIX conference on Hot Topics in Cloud Computing*. USENIX Association. 2014, pp. 4–4.

[61] L. Shi, H. Chen, J. Sun, and K. Li. "vCUDA: GPU-accelerated high-performance computing in virtual machines". *Computers, IEEE Transactions on* 61.6 (2012), pp. 804–816.

[62] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors". *ACM SIGOPS Operating Systems Review*. Vol. 41. 3. ACM. 2007, pp. 275–287.

[63] R. Thion and D. Le Metayer. "FLAVOR: A Formal Language for a Posteriori Verification of Legal Rules". *Policies for Distributed Systems and Networks (POLICY), 2011 IEEE International Symposium on*. June 2011, pp. 1–8. DOI: 10.1109/POLICY.2011.26.

[64] K. Tian, Y. Dong, and D. Cowperthwaite. "A full GPU virtualization solution with mediated pass-through". *Proc. USENIX ATC*. 2014.

[65] R. Trapero, N. Suri, and TUDA. *A Common Reference Model to describe, promote and support the uptake of SLAs*. Tech. rep. May 2016.

[66] VMware, Inc. *VProbes Programming Reference*. Tech. rep. 2011.

[67] M. Weiser. "Hot topics-ubiquitous computing". *Computer* 26.10 (Oct. 1993), pp. 71–72. DOI: 10.1109/2.237456.

[68] F. Westphal, S. Axelsson, C. Neuhaus, and A. Polze. "VMI-PL: A monitoring language for virtual platforms using virtual machine introspection". *Digital Investigation* 11.S-2 (2014), S85–S94.

[69] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose. "Performance evaluation of container-based virtualization for high performance computing environments". *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. IEEE. 2013, pp. 233–240.

[70] Xen Project. *The Next Generation Cloud: The Rise of the Unikernel*. Tech. rep. http://xenproject.org, 2015.

[71] S. Xiao, P. Balaji, Q. Zhu, R. Thakur, S. Coghlan, H. Lin, G. Wen, J. Hong, and W.-c. Feng. "VOCL: An Optimized Environment for Transparent Virtualization of Graphics Processing Units". *Proceedings of 1st Innovative Parallel Computing (InPar)*. 2012, pp. 1–12.

[72] J. Yang, K. Yessenov, and A. Solar-Lezama. "A Language for Automatically Enforcing Privacy Policies". *Proc. 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '12. ACM, 2012, pp. 85–96. DOI: 10.1145/2103656.2103669.

[73] E. Zhai, G. D. Cummings, and Y. Dong. "Live migration with pass-through device for Linux VM". *OLS'08: The 2008 Ottawa Linux Symposium*. 2008, pp. 261–268.