A Privacy Preserving System for Cloud Computing

Ulrich Greveler, Benjamin Justus, Dennis Loehr
Dep. of Electrical Engineering and Computer Science
Münster University of Applied Sciences
Steinfurt, Germany
Email: {greveler, benjamin.justus, loehr}@fh-muenster.de

***Abstract*—Cloud computing is changing the way that organizations manage their data, due to its robustness, low cost and ubiquitous nature. Privacy concerns arise whenever sensitive data is outsourced to the cloud. This paper introduces a cloud database storage architecture that prevents the local administrator as well as the cloud administrator to learn about the outsourced database content. Moreover, machine readable rights expressions are used in order to limit users of the database to a need-to-know basis. These limitations are not changeable by administrators after the database related application is launched, since a new role of *rights editors* is defined once an application is launced. Furthermore, trusted computing is applied to bind cryptographic key information to trusted states. By limiting the necessary trust in both corporate as well as external administrators and service providers, we counteract the often criticized privacy and confidentiality risks of corporate cloud computing.**

***Keywords*-Access control, Data security, Software architecture, Outsourcing, Privacy.**

I. Introduction

While the storage of corporate data on remote servers is not a new development, current expansion of cloud computing justifies a more careful look at its actual consequences involving privacy and confidentiality issues.

We introduce in this paper a secure privacy preserving cloud database storage architecture. We focus on the *Software as a Service* [1] kind of utility computing model. The proposed architecture addresses the important security question: 'To what extent, an organization has to trust client users, system administrators and service providers?'

Remote access to corporate networks is already an essential aspect of the corporate work environment. Client systems are used by employees both inside the corporate Intranet, and remotely from home. In both scenarios, applications require effective requests and use of Cloud-based computing resources on the fly.

Privacy concerns arise whenever sensitive data is outsourced to the cloud. By using encryption, the cloud server (i.e. its administrator) is prevented from learning content in the outsourced databases. But how can we also prevent a local administrator from learning the database content. And how can we avoid scenarios such as: employees using cloud applications may learn more than it is necessary to perform their respective duties?

As an illustration, an organization may want to specify rules limiting request-per-day for call center employees to 100 client contacts. Such restrictions prevent download of the whole (customer) database. Our contribution in this paper is a system architecture that allows sufficient and flexible restriction writing. And in doing so, local administrators as well as cloud administrators are not able to change the access rules after an application is launched.

A. Related Work

Maheshwari et al. [2] describe a way of securing larger untrusted storage such as a hard-disk. This solution needs a small secure storage for decryption keys. A practical solution based on this approach is the Microsoft Bitlocker [3], which uses TPM (Trusted Platform Module[4]) chip as a storage for decryption keys. Our system is based on Linux operating system. We use the TPM sealing function with a key-file for dm-crypt with LUKS[5].

The de facto standard for access control languages at present is the eXtensible Access Control Markup Language, or short XACML[6]. Using XACML, one is able to realize a fine grained and flexible access control policy. Because XACML is XML based, we can embed XML-Signatures [7] to secure submission of new access-control-rule-files.

Kubiatowicz et al. [8] describe an architecture for encrypted persistent storage in the Cloud called *OceanStore*; Sadeghi et al. [9] describe a technique through usage of a tamper-proof hardware token. This techniques applies to a cloud service provider which are based on trusted computing platform. Another project is from Pearson et al. ([10], [11]) that combines some of the previous techniques. The key idea is the creation of a *Privacy Manager*, and a role represented by a Trusted Platform Module. The *PrimeLife* project [12] has developed a system that combines the *PRIME* privacy technologies with XACML access control. However, all these systems do not address safeguards against internal attackers.

While in general an employee cannot access all the data stored on the cloud, there often exists in a company at least one privileged role with unlimited administrative accesses. Our proposed cloud database storage architecture provides data privacy without the need to trust corporate administrators as well as external cloud administrators. In the rest of the

paper, we shall frequently bring up the following example as a way of describing the practical benefits of our system: A company have call-in client contacts organized over an external call-center. We would like to limit the access-count of readable client files towards external employees in order to reduce the risk of losing sensitive data.

B. Outline

This paper is organized as follows. In section II, we give an overview over the technologies we use in the system. Section III and IV describes the system components in details. In section V, we discuss involved security factors and section VI shows how we may improve the system in the future.

II. Background on related technologies

In this subsection we give an overview of the technologies used in the proposed system. These are in particular the *Extensible Markup Language* (XML) based standards XML-Encryption, XML-Digital-Signatures and XACML. Moreover, we utilize the Trusted Platform Module (TPM) and use homomorphic encryption.

1) XML: The *Extensible Markup Language* (XML) is designed for Internet Application uses. Generally, the specification [13] describes a way to encode information in a plain text file. The file format is both machine and human readable. XML document encoding was designed in 1996[1] by the World Wide Web Consortium[14].

2) XML-Encryption: The *Extensible Markup Language Encryption Syntax and Processing* [15] is an extension to the XML syntax. The syntax makes embedding of encryption within XML files possible. The XML-encryption includes the following features: encryption of a whole XML document, encryption of a single element and the content of an element. Furthermore, encryption upon encryption is also possible. Geuer-Pollmann[16] introduced a way of hiding elements by position shifting in a XML document.

3) XML-Signature: The standard *XML Signature Syntax and Processing* [7] is maintained by World Wide Web Consortium. It allows signature embedding within a XML file.

4) XACML: The *eXtensible Access Control Markup Language* (XACML) is developed by the OASIS [17] committee. XACML is an XML based access control language [6]. It defines a policy format, also a request and response message format. Furthermore, XACML allows users to define policies, combination of policies and attribute based restrictions which we can use to to realize our system and define request limits in addition to the four-eyes principle XACML file.

[1]http://www.w3.org/TR/WD-xml-961114.html

Table I
Role Overview

Role Name	Identifier	Rights
System-Administrator	A	backup encrypted data & setup Encryption Proxys
XACML editor	$E_1, E_2, ..., E_n$	develop and activate new access rules
Developer	D	create new requests
Employee/Worker	$W_1, W_2, ..., W_m$	access a limited amount of data sets

5) Trusted Platform Module: The *Trusted Computing Group* (TCG) proposed a trust model where each device is equipped with a hardware root-of-trust associated with the platform that can measure integrity metrics and may confirm these metrics to other parties. Regarding PC or server hardware the hardware chip is called the Trusted Platform Module (TPM) and the process of reporting the integrity of a platform is known as *remote attestation*. TPM has a secure non-volatile memory to store RSA-keys and can bind a file to a specific hardware state. Moreover, a software based Trusted Platform Module [18] was proposed. In such a way the *Infrastructure as a Service* (IaaS) systems may offer TPM-like functions in a virtual environment.

III. Proposed System Concept

We aim to build a secure system that can fend off both external and internal attackers[2]. Many previous work deal with issues related to external attackers. Our system combines many existing techniques which we explain in the following sections. We summarize the interaction roles of the system in Table I.

Data availability has a very high priority in any company operations. In our system, all data are stored encrypted. The backup of the database is performed regularly by the cloud service, in addition we require a backup of the Encryption Proxies with the corresponding decryption keys for the system integrity.

We automate the backup procedure for Encryption Proxies by establishing system integrity first, then exchanging the decryption keys over a secure channel. All session keys are TPM sealed. By comparing specific PCR values, we are able to attest the integrity between identical hardware. And only if both Encryption Proxies have the same state (same hardware, software, XACML file and known database services), the exchange of their key material can take place.

Using XACML, our system not only can limit the number of queries q by an employee W_j, it is also possible to setup a fine grained access control structure. XACML editors should follow the what-need-to-know principle. With respect to confidentiality, employees must only access what they

[2]Currently we work on a prototype for our System and make this prototype available at www.daprim.de.

need in their jobs (so system administrators do not need to access the productive database. In particular, we want to avoid the possibilities of employees copying the entire database.

With this in mind, the process of integrating a new function in the system is as follows. We illustrate this by considering our old example: design a new per-salesman business volume request for the call center. First a developer D has to design the new request and sends it to an editor. Next a XACML trusted editor E_i (among all editors $M = \{E_1, E_2, ..., E_n\}$) integrates the request and sets up the access rights for the new request. After integration and setting up the new rules, the new XACML file is signed and sent to another editor $E_i \neq E_j$. The second editor then checks the XACML file and after verification signs the file next.

The new access control file can now be distributed over the Encryption Proxies. An Encryption Proxy accepts a new XACML file on the condition: the file is signed by two (or more) editors and all their signatures are good. If the file is not proper, or correctly signed, the system rejects the new file and falls back to the current rights (for availability reasons).

The system administrator A has the role of setting up an environment, and a base system for the encryption proxy. At this stage, the system administrators have privileges of doing anything they like in the system. The system afterwards is not anymore modifiable except for those unmeasured configurations. This is the post-system setup stage, in which system-administrators A have the tasks of cloning new encryption proxies (same hardware, same software and same configuration) and managing the configuration files. The system administrators have for each proxy an admin-panel, so he can submit new (valid-signed) XACML-files, change the administration credentials (for the admin-panel), and to edit lists of used cloud databases.

A. System Architecture Details

The productive database is stored in the cloud, and our proposed architecture aims to protect the content of the database. The system consists of a *Data Management*, *Encryption Proxy* and a *User Interface*. The cloud provides database services and plays the role of *data management*. The size of the cloud is not restricted, for example that our system would work on a private cloud or single database server.

The content of the productive database is encrypted. The user privacy and confidentiality is achieved because the cloud have no access to the original database content. The clients are not bound to a single Encryption Proxy, and it is also possible to use a load balancer to enhance the performance of the system.

The productive database includes a meta data table where meta information of each user's transactions are stored. For example, the table could contain a counter which holds the number of service request by a client. This information can be used in the case, when the system is designed to limit the number of service request by a single or a group of clients. A user is not bound to a single encryption proxy, and a load balancer can be used to distribute work load over several proxies. The meta-table has to be signed for protection against replay attacks from cloud administrators. The signing operation can be achieved by the TPM Quote functionalities. When signing, the TPM quote contains information about the hardware state and the status of request tables.

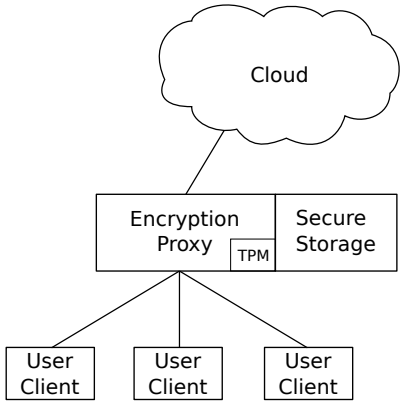

Figure 1. System Overview

The encryption proxy is the key part of the system. It provides user access to the (unencrypted) data. The encryption proxy acts as an intermediary between cloud and users (see figure 1). An encryption proxy also serves secure data storage. The secure storage is achieved by the use of a full disk encryption with a TPM protected key-file. The storage is dedicated to the use of data management. All other information are encrypted and stored in external data storage ('in the Cloud').

IV. Encryption Proxy

Figure 2 gives a detailed view of the *encryption proxy*. A clarification of the notations we use in this section: $[x]_\tau$, $\{x\}_\tau$, and $[[x]]_\tau$ mean respectively asymmetric encryption of x, signing of x, and homomorphic encryption[19], [20] of x using the key τ.

The Linux-based encryption proxy has three parts: a user-engine, a rule-engine, and a secure storage. TPM is used to provide secure data storage. The TPM measures and stores the system BIOS boot in PCR 0-7, IMA Grub in PCR 8-9 and 12-14 and IMA-kernel in PCR 10. PCR 10 reflects any program alteration in the system. In addition, the system provides no login shell.

TPM measurements by themselves do not stop the system from insecure operations. The TPM sealing function binds data to a secure state. In order to carry out the sealing

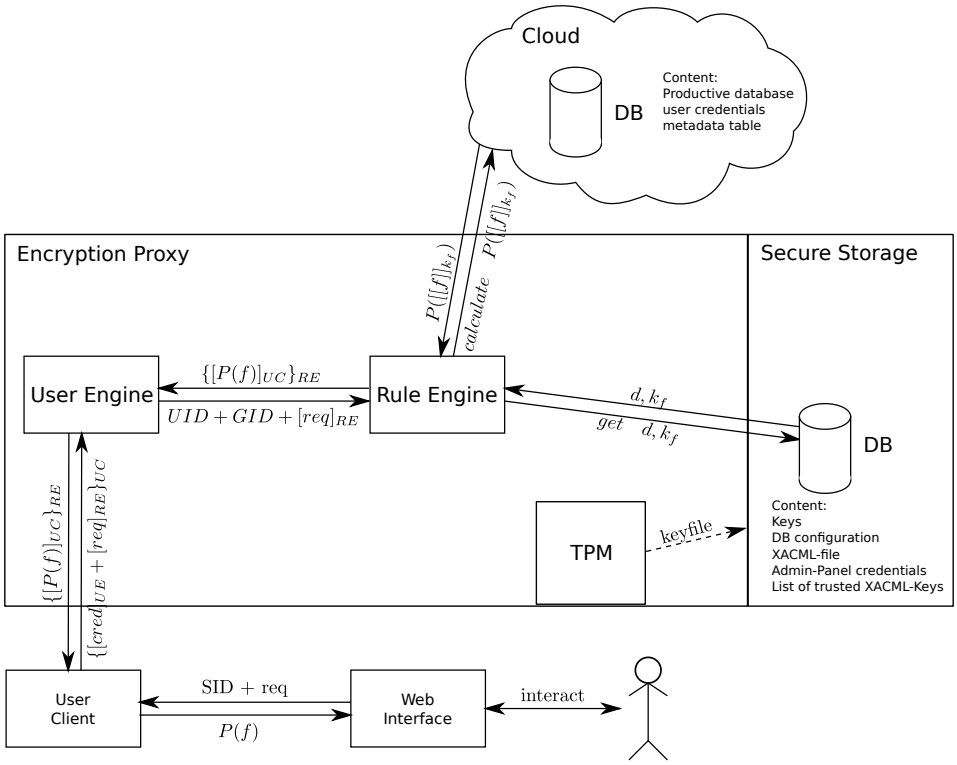

Figure 2. Encryption Proxy View

functions, the TPM binds on the states which are represented by all the PCRs. Since Boot-up decryption of the whole secure storage takes long, we propose transparent disk encryption *dm-crypt* with *LUKS*[5]. This allows decryption only on-demand and reduces time consumption while the system is booting up (in contrast to desealing the database on start-up).

In addition, we need an access control language such as XACML for the rule-engine. XACML enables us to have complex access rights together with XML-Signatures. We can specify rules to limit the daily requests for call center employees (as stated in the introduction) and prevent download of the whole database. As an illustration, we added in figure 3 a readable XACML document with two signatures, which are similar to those used in our system.

The rule-engine uses its secure storage to store data management information (e. g. realize the organization of the data within the cloud). In the rule-engine, the first request fires the decryption signal to TPM to access its secure storage. This will only succeed if no PCR has changed since the creation of the database. Only the content of the secure-storage is changeable within the encryption proxy. Furthermore only the XACML-file, the admin-panel credentials, the list of trusted XACML-Editors M and the credentials for the used cloud databases are changeable.

Figure 2 shows a detailed overview of a user request. A user here interacts with the webinterface of the system. If the webinterface needs some information from the system, it sends a request req together with the session ID SID. Communication to the user-client is over a secure channel (TLS).

User-client UC afterwords creates a XML-RPC request to the user-engine UE. This XML-RPC consists of the credentials $cred$ and the request from webinterface. To secure the request, all parts are encrypted (with XML-ENC) and the whole request $\{...\}_{UC}$ is signed (XML-DSIG) by the UC. The credentials are encrypted with the user-engine key $[cred]_{UE}$ consequently the req from the webinterface is encrypted with the rule-engine RE key $[req]_{RE}$.

After receiving a request, the user-engine checks the signature. If the signature is genuine, the system will decrypt the user credentials $cred$. The user-engine next checks the credentials. The user-engine appends the UserID UID and the GroupID GID to encrypted requests and forwards it to the rule-engine.

Next the rule-engine looks up the access control file for allowed functions related to UID (or GID). When a function P is found, the rule-engine also ascertained the used fields f. With that the RE search in the secure-storage for the used database d and the decryption keys k_f. Next the RE calculates the computation request for the database $P(f)$ and sends the request to the cloud database.

```
<?xml version="1.0" encoding="UTF-8">
<Policy xmlns=... PolicyId="paper"
 RuleCombiningAlgId="urn:oasis:...">
  <Description>
    This is an example XACML-file
  </Description>
  <Target>
   <Subjects> <AnySubject/> </Subjects>
   <Resources> <AnyResource/> </Resources>
   <Actions> <AnyAction/> </Actions>
  <Target>
   ...
  <Rule RuleId="..." Effect="Permit">
   <Description>
     This is an example rule
   </Description>
   <Target>
    <Subjects> <AnySubject/> </Subjects>
    <Resources> <AnyResource/> </Resources>
    <Actions> <AnyAction/> </Actions>
   </Target>
   <Condition>
    ...
   </Condition>
  </Rule>
  ...
</Policy>
<Signature id="sig1">
 <SignedInfo>...</SignedInfo>
 <SignatureValue>...<SignatureValue>
 <KeyInfo>...<KeyInfo>
</Signature>
<Signature id="sig2">
 <SignedInfo>...</SignedInfo>
 <SignatureValue>...<SignatureValue>
 <KeyInfo>...<KeyInfo>
</Signature>
```

Figure 3. Structure of an XACML file with two signatures.

To fulfill the request, the cloud service calculates now the function $P([[f]]_{k_f})$ and sends it back to the RE. RE uses now the Key k_f to decrypt the result and generate a XML-RPC response. The calculated result $P(f)$ is then XML-encrypted with the key of the UC and signed by the RE $\{[P(f)]_{UC}\}_{RE}$. After that it will be sent to the UE that pass through this response to the UC. User-client checks the signature of the response and decrypts it. To the end the response is sent to the web interface. So the user has his new responses.

V. Conclusion

Data security and privacy is one of the biggest challenges in Cloud Computing. Cloud data must be protected not only against external attackers, but also currupt insiders. Our proposed system follows the *information-centric* approach which aims to make cloud data self-intelligent. In this approach, cloud data are encrypted and packaged with a usage policy. The data when accessed will consult its policy, create a virtualization environment, and attempt to assess the trustworthiness of the data environment (using Trusted Computing).

Our contribution in this paper is a privacy preserving database storage architecture. Such a system is often desirable in a corporate setting, in which database containing sensitive information need to be protected not only against external adminstrators, service providers, but also local adminstrators. Our system also allows via machine readable rights expressions depth control over information that is alloted to a particular user.

VI. Future Work

While vulnerabilities regarding the confidentiality of data outsourced to the cloud can easily be addressed through the adoption of industry standard encryption technologies, the creation of complex machine readable access rights to the decryption keys becomes a challenging problem. The syntax of XML-based rights expressions is complicated and obscure when the user-related conditions become sophisticated. The functionality of being able to handle a wide variety of possible access scenarios is typically built into any rights expression language, but it is often difficult to cleanly partition out those subsets needed by a particular privacy preserving application. How to efficiently generate rights expressions reflecting the requirements of an organization and being secure at the same time becomes a future challange.

References

[1] H. Hacigümüş, B. Iyer, and S. Mehrotra, "Providing database as a service," in *Proceedings of the International Conference on Data Engineering*, Los Alamitos, CA, USA, 2002.

[2] U. Maheshwari, R. Vingralek, and W. Shapiro, "How to build a trusted database system on untrusted storage," in *Proceedings of the 4th USENIX Symposium on Operating System Design and Implementation*, Berkeley, CA, USA, 2000.

[3] N. Ferguson, "AES-CBC+ Elephant diffuser A Disk Encryption Algorithm for Windows Vista," *Microsoft*, 2006. [Online]. Available: http://download.microsoft.com/download/0/2/3/0238acaf-d3bf-4a6d-b3d6-0a0be4bbb36e/bitlockercipher200608.pdf

[4] Trusted Computing Group, *TPM Main Specification Version 1.2, Revision 116*, 2011. [Online]. Available: http://www.trustedcomputinggroup.org/resources/-tpm_main_specification

[5] C. Fruhwirth, "LUKS On-Disk Format Specification Version 1.1," 2005. [Online]. Available: http://code.google.com/p/cryptsetup/

[6] *OASIS XACML committee website*, 2011. [Online]. Available: https://www.oasis-open.org/committees/xacml/

[7] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, *XML Signature Syntax and Processing (Second Edition)*, D. Eastlake, J. Reagle, D. Solo, F. Hirsch, and T. Roessler, Eds., 2008. [Online]. Available: http://www.w3.org/TR/xmldsig-core/

[8] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, "Oceanstore: an architecture for global-scale persistent storage," *SIGOPS Oper. Syst. Rev.*, vol. 34, pp. 190–201, November 2000.

[9] A.-R. Sadeghi, T. Schneider, and M. Winandy, "Token-based cloud computing," in *Trust and Trustworthy Computing*, A. Acquisti, S. Smith, and A.-R. Sadeghi, Eds. Springer Berlin / Heidelberg, 2010, vol. 6101, pp. 417–429.

[10] S. Pearson, Y. Shen, and M. Mowbray, "A privacy manager for cloud computing," in *Cloud Computing*. Springer Berlin / Heidelberg, 2009, pp. 90–106.

[11] M. Mowbray, S. Pearson, and Y. Shen, "Enhancing privacy in cloud computing via policy-based obfuscation." Springer Berlin / Heidelberg, 2010, pp. 1–25.

[12] C. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, and P. Samarati, "An XACML-based privacy-centered access control system," in *Proceedings of the first ACM workshop on Information security governance*. New York, NY, USA: ACM, 2009, pp. 49–58.

[13] T. Bray, J. Paoli, E. Maler, F. Yergeau, J. Cowan, and C. M. Sperberg-McQueen, Eds., *Extensible Markup Language (XML) 1.1 (Second Edition)*, 2006. [Online]. Available: http://www.w3.org/TR/xml11/

[14] *World Wide Web Consortium (W3C)*, 2011. [Online]. Available: http://www.w3.org/

[15] T. Imamura, B. Dillaway, and E. Simon, *XML Encryption Syntax and Processing*, D. Eastlake and J. Reagle, Eds., 2002. [Online]. Available: http://www.w3.org/TR/xmlenc-core/

[16] C. Geuer-Pollmann, "Xml pool encryption," in *Proceedings of the 2002 ACM workshop on XML security*. New York, NY, USA: ACM, 2002, pp. 1–9.

[17] *Organization for the Advancement of Structured Information Standards(OASIS)*, 2011. [Online]. Available: http://www.oasis-open.org/

[18] M. Strasser and H. Stamer, "A software-based trusted platform module emulator," in *Trusted Computing - Challenges and Applications*. Springer Berlin / Heidelberg, 2008, pp. 33–47.

[19] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 2009, pp. 169–178.

[20] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology EUROCRYPT 2010*, H. Gilbert, Ed. Springer Berlin / Heidelberg, 2010, vol. 6110, pp. 24–43.