2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing

HPISecure: Towards Data Confidentiality in Cloud Applications

Eyad Saleh
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
eyad.saleh@hpi.uni-potsdam.de

Christoph Meinel
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
christoph.meinel@hpi.uni-potsdam.de

***Abstract*—Cloud computing has emerged over the last years as a new model of delivering computation to consumers. Due to the nature of the cloud, where the data and the computation are beyond the control of the user, data privacy and security becomes a vital factor in this new paradigm. Several research studies reported that security and privacy is cited as the biggest concern in adopting cloud computing. In this paper, we present *HPISecure*, a software prototype designed to facilitate the process of securing data stored on the cloud. *HPISecure* allows the user to store an encrypted version of his data on the cloud without breaking the functionality of the application. *HPISecure* intercepts the HTTP request/response objects, encrypt data before transmitting to the cloud, and decrypt data received back from the server. We have successfully tested *HPISecure* with Google Docs and Google Calender. Future work includes extending *HPISecure* to work with enterprise applications.**

I. Introduction

Software-as-a-Service (SaaS) has been growing rapidly over the last years and seems to be a promising software delivery model [1]. Gartner reported that SaaS is expected to have a healthy growth through 2015, where worldwide revenue is projected to reach $22 billion [2]. SaaS provides major advantages to both service providers as well as consumers. Service providers can provision a single set of hardware to host their applications and manage hundreds of clients (tenants). They can easily install and maintain their software. As for the consumers, they can use the application anywhere and anytime, they are relieved from maintaining and upgrading the software (on-premises scenario), and benefit from cost reduction by following the pay-as-you-go model [3].

Although SaaS offers several advantages to both service providers and users, such as the reduction of *Total Cost of Ownership* (TCO), better scalability, and better resource utilization, the users are still concerned about the security and privacy of their data. Privacy concerns exist wherever information about individual or organizations is processed and stored. Improper disclosure of information can be the cause for privacy issues. In 2009, Forrester Research reported that *Privacy and Security* has been selected by IT professionals as the primary barrier for not widely adopting the cloud computing.

Data encryption is a common approach to protect the confidentiality of user's data during transmission and storage [6], [7]. As for computation, a fully homomorphic encryption scheme has been introduced by [8] as a result of joint efforts between Stanford and IBM. This scheme supports computation over encrypted data. However, we believe that such a technique is still in the early stages of development, and would require huge extra cost. Another approach can be referred to as *information disassociation* [18], where the information is separated into parts, these parts are stored in geographically-distributed locations, therefore any party involved can not benefit from the data it hosts. In our work, we offer a hybrid-approach, where we combine the usage of encryption and disassociation to increase the level of confidentiality the user is looking for. Recent approaches such as *CloudProtect* [19] and *Silverline* [20] are closely related to *HPISecure*. Both of them support keeping the data at the server-side confidential by encryption in away that is transparent to the application. However, both of them requires additional software or configuration to be made on the client machine, which prevents the user from using other computers to access his data (which we try to avoid), and hence, violates the basic concept behind the cloud.

The contribution of this paper is threefold: First, we introduce a hybrid-approach to achieve data confidentiality without breaking the functionality of the application. Second, we present a technique to distribute the data on multiple storage providers to increase the level of security. Third, we implement our approach as an HTTP proxy that could be installed on the client's machine, and evaluate it against Google Docs and Google Calendar.

The rest of this paper is organized as the following: We provide a background in Section II. An overview of *HPISecure* and the details of its components is presented in Section III. In Section IV, we present some experiments. Section V outlines the limitations of our approach. Finally, we discusses the related work and concludes the paper in Section VI and VII respectively.

II. Background

The definition of privacy varies widely among countries, organizations, cultures, and even individuals. However, it shares (to some extent) common basics. Privacy can be defined as *the ability or wish of an individual or group to remain unidentified to the public society*. Another definition that is becoming popular is *The rights and obligations of individuals and organizations with respect to the collection, use, retention, and disclosure of personal information*. This definition has been provided by the American Institute of Certified Public

978-0-7695-4996-5/13 $26.00 © 2013 IEEE
DOI 10.1109/CCGrid.2013.109

Accountants (AICPA) and the Canadian Institute of Chartered Accountants (CICA) in the Generally Accepted Privacy Principles (GAPP) standard [4].

Online inquiries about individuals and organizations have grown dramatically over the last decade. Social-networking websites have millions of users registered, the information of those users are subject to privacy violation. For instance, Microsoft reported that 75 percent of the US human resource professionals are using the internet to inquire about their candidates. In addition, 70 percent of the recruiters have rejected candidates based on online resources [5].

Data privacy is a critical issue in several domains, such as healthcare, financial records, criminal records, residence and geographic records, etc. In addition to that, particularly in SaaS, the storage of data and the computation is beyond the control of the user, therefore the user needs to make sure that high-level standards of security and privacy are implemented to keep his data confidential.

Although service providers claim that their data centers are secured by implementing complex security protocols and enforcing privacy best-practices, actual deployment does not really reflect this. For instance, Amazon's S3 was interrupted twice in 2009. Another issue in 2009 was with Google Mail, where a security vulnerability led to serious leakage of user private information. Microsoft fell into such issues also when the Azure platform had an outage for 22 hours. And therefore, our efforts in this paper try to propose a solution to such a critical issue.

III. Overview of HPISecure

The overall goal of *HPISecure* is to improve the confidentiality of users' data stored on the cloud. We propose an end-to-end encryption approach using public-key cryptography to secure the data. Currently, we focus on the documents-based applications, such as Google Docs. In our future work, we will study the database-based applications, such as Customer-Relationship Management (CRM) and Human Resource Management (HRM) applications. We implemented *HPISecure* on top of Fiddler [11] as an HTTP proxy installed on the client machine, where the Request/Response objects of the HTTP protocol are intercepted, configured encryption/decryption algorithms are applied, and existing exceptions (if there is any) are enforced.

Currently, there are several applications that offer securing the content on the cloud by utilizing cryptographic techniques [9], [10]. However, such applications limit the user to use his personal computer whenever he needs to access his data, because the application, the encryption/decryption keys, as well as other information is stored on his machine along with the application. And hence, he can not use other computers to access his data, which violates the basic concept behind the cloud (access your date anywhere and anytime). Therefore, our approach overcomes this limitation by introducing the concept of the *Facilitator* as shown in Figure 1.

The *Facilitator* here could be the company that the user works for, it could be another cloud provider, or it could be a USB device. The idea of the *Facilitator* is simple, we need to relieve the client machine from storing the cryptographic keys and other related data, and move this stuff to the facilitator. In this case, the client machine will be transparent to our approach, i.e., the user can use any computer to securely access his data, not only the machine that he used for the first time.

If the user works for a company, the *Facilitator* would be the internal web server of the company, where *HPISecure* will be installed. When the client try to create/save his data on the cloud for the first time, the request will be routed through the company's internal server, where the request will be intercepted, the data will be encrypt according to the preferences configured in *HPISecure* for this user, then proceed with the request to store the data securely in the cloud. For the upcoming requests when the user browse his data, the request for fetching the data will be also routed through the company's internal server, when the encrypted data retrieved, it will be intercepted, decrypted according to the user preferences by *HPISecure*, then sent to the user machine.

Individuals can use the same concept, but the *Facilitator* in this case refers to a third-party cloud provider, where the keys and related information will be stored. The same process mentioned earlier will be applied here. Finally, if the user doesn't belong to an organization (i.e., individual) and does not prefer to use third-party providers, the USB device is the choice. Using the USB device allows the user to store the cryptographic keys, configuration files, and any related information on a USB device, whenever he needs to use a new computer, all he requires is installing *HPISecure*, plug the USB, retrieve the keys and configuration files, then securely send/receive his data.

A. Archeticture of HPISecure

The architecture of *HPISecure* is shown in Figure 1. The main components are: *Encryption Manager*, *HTTP Parser*, *Algorithms Manager*, *Keys Manager*, *Database*, *Exceptions Handler*, *Distribution Manager*, *Controller*, and *Facilitator*. To better explain *HPISecure*, we will describe its components using a use case as detailed below.

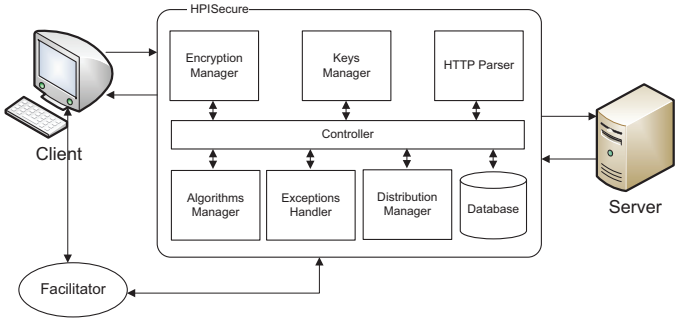

Fig. 1. HPISecure Architecture

1) Encryption Manager: Google Docs is widely used to create new documents or modify existing ones. That is why we select it to be our main use case. *HPISecure* intercept the Request/Response HTTP objects to encrypt/decrypt the content before sending/receiving it from the server. For example, when

the user creates a new document, the following steps are accomplished before saving the document to the server:

- *HPISecure* calls the *HTTP Parser* to read the content of the Request object (i.e., the content of the document)
- In case of Google Docs, we split the content into two parts according to the index of the characters. All characters that have odd index are stored in part one, while the others (i.e., characters of even indexes) are stored in part two.
- The *Algorithms Manager* randomly select one of the standard encryption algorithms that the user configure to use by *HPISecure*, such as RSA or AES256.
- Two different keys are selected from a pre-defined group of keys by the *Keys Manager*. Each one is assigned to the newly created parts that have been shown in Figure 2.
- If there are exceptions created by the user, the *Exceptions Handler* is responsible for applying them.
- The Distribution Manager is used to handle how the storage of the document after encryption looks like. For instance, is it going to be stored only on Google's servers, or each part of the document (the newly created parts) should be stored in two different locations, e.g., one on Google's servers while the other on Amazon S3.
- Finally, the *Encryption Manager* combine all the above steps, store these details in the database for future use, and then issues an encrypted content, then post it to the server(s) according to the specifications of the *Distribution Manager*.
- When the user request to read the document, the *Encryption Manager* follows the same steps above, but to decrypt the content (instead of encrypt) and displays it on the user's screen.

2) HTTP Parser: The main role of the *HTTP Parser* is intercepting the Request/Response objects between the client and the server, extract the required information from the headers and the body of the object, then pass these information to the *Controller*. In some cases, additional headers might be added, such as the date. The *Controller* will also pass the encrypted data back to the *HTTP Parser* to post it to the server.

Fiddler [11] offers the possibility of intercepting requests before and after sending them to the server. It also offers the same for the response objects. Thus, by utilizing Fiddler, we are able to intercept and manipulate request/response objects when required. Furthermore, Fiddler offers the entire request/response objects and their data as a .NET objects, that is accessible to any application that extends Fiddler, so extending Fiddler allows us to read all headers and body content of the request/response objects, add new content, change exiting content, and delete unwanted data.

3) Algorithms Manager: This component is simply responsible for selecting randomly one of the encryption algorithms from the user's pre-defined list. Encryption algorithms here refer to any standard algorithm, such as RSA or AES256. Since encryption itself is not a focus-point in this paper, we will not discuss the details of implementing them. This selected algorithm is used to encrypt the current document. This indicates that the same document might be encrypted using different algorithms, and therefore, storing these information in an internal database for further usage is necessary, and this is the role of the *Database* component.

When an encryption algorithm is selected to be used, HPISecure stores all related information in the database, such as document title, creation date, modification date, user-id, algorithm name, encryption key, etc. This information will be called back to decrypt the document when the user browse it again. Worth to mention that the database is neither stored on the client machine, nor on the provider side, the *Facilitator* is used to achieve this goal. The details of the *Facilitator* has been described in Section III.

4) Keys Manager: Two major types of encryption could be used to protect data, symmetric-key cryptography and public-key cryptography. symmetric-key uses the same key for encrypting and decrypting the data, while the public-key uses two keys, a public one for encrypting, and a private key for decrypting the data. *HPISecure* uses public-key cryptography. However, to make it very hard for malicious users to uncover the private key of the user, the *Key Manager* randomly select a key among a group of private keys that the user use, thus, the same document might be encrypted using different keys. As a result, if a malicious user suddenly get access to the user's private key, this does not explicitly means that he can access his documents, because every document may be encrypted using different keys.

Unfortunately, our approach limits the sharing and collaboration of encrypted documents. However, the most obvious solution for this limitation is to use symmetric-key cryptography, and share the encryption key with all parties. Although this sound applicable, it does not reflect a high-level of security. Therefore, one of our extensions to *HPISecure* would focus on what we call the *group signature*, where we use one encryption key for a group of users whom sharing a certain document. Consequently, the encryption will be group-based instead of document-based, which increases the level of security and reduces the number of keys, and hence, reduces the complexity of managing them.

5) Exceptions Handler: It is obvious that protecting data using cryptography involving extra overhead that grows with the desired level of security. Thus, allowing the user to trade-off the level of security for performance is useful. *Exceptions Handler* facilitate this purpose by allowing the users to define exceptions and actions. Actions might vary from no encryption to strong encryption. For instance, an exception might be (*if the document contains the word 'price'*) then the action should be '*strong encryption*'. While another exception might be (*if the title of the document has the word 'John'*) then *no encryption* is needed. Strong encryption means that the document is splitted into two parts, encrypted using a complex encryption algorithm, and then distributed on different locations. These exceptions are stored in the database and can be modified as needed.

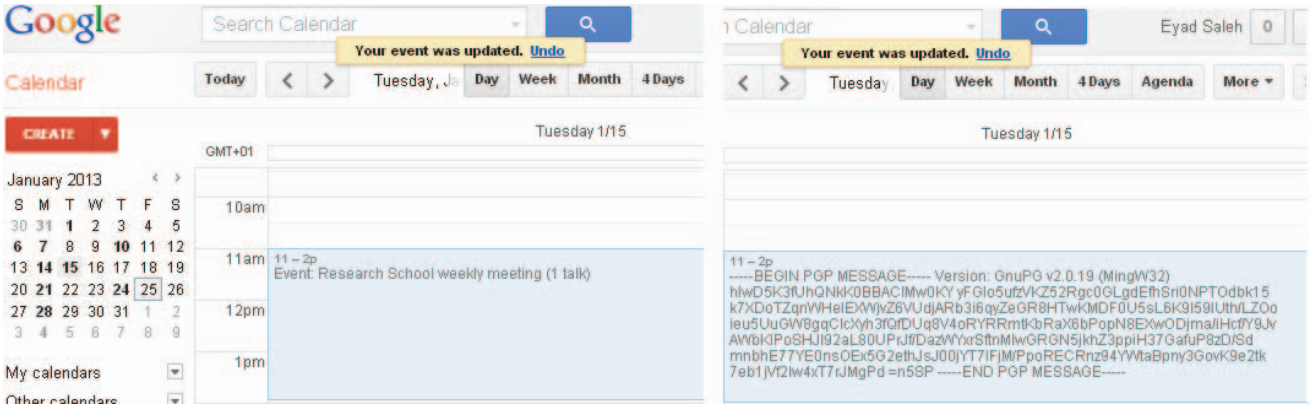

Fig. 2. An Example of a Calendar Event Before and After Encryption

IV. Experiments

We develop *HPISecure* as a proof-of-concept to validate our approach. We tested our approach against Google Docs and Google Calendar. Figure 3 shows an example of a Google Calender's event before and after encryption.

Google Calendar. The user opens Google Calender and create a new event, *HPISecure* intercept the event and encrypt the data, then send it to the server. Thus, the data will be securely stored at the server. When the user browse the event again, the encrypted data will be retrieved by the browser, *HPISecure* decrypt the data (on the user's machine) according to the private-key that matches the public-key that is used to encrypt the data, then displays the plain text to the user. Required meta-data to handle the encryption/decryption is stored in the database, such as IDs of the events, the ID of the key used for encryption, etc.

Google Docs. There are basically two options to work with *Google Docs*, either you create new file on the fly, or upload a file from your local drive. As a proof-of-concept, we focus only on the first option (on the fly). As with Google Calendar, *HPISecure* reads the content of the file, split it into two parts (as detailed in section II), then encrypt them. A call to the distribution manager to decide how to store the encrypted parts. We incorporate Amazon's .NET SDK for S3 in *HPISecure* to create, manage, and delete objects in Amazon S3.

V. Limitations of HPISecure

Enterprise Applications. The main target of *HPISecure* is the document-based applications. Encrypting documents is much easier than structured data, such as database records. Therefore, we started with it, and part of our future work is to extend *HPISecure* to work on enterprise applications, such as Customer Relationship Management (CRM) and Human Resource Management (HRM).

Availability. Securing data on the cloud involve several factors, such as cost, availability, performance, etc. Although we provide a distribution technique to achieve higher level of security and performance, availability concerns remains in question.

Desktop Implementation. *HPISecure* is currently implemented as a desktop application, which violates the concept of mobility available in the cloud. Thus, a web-based implementation of *HPISecure* is needed. We will consider this also in our future work.

VI. Related Work

Protecting users' data is an essential task in current systems. Researchers are proposing approaches and solutions to maximize the confidentiality of users' data. Social networks is considered as an interesting area to study the impact of privacy issues on. FlyByNight[12] and Persona[13] are mainly designed to work with social networks, such as Facebook. They propose to store an encrypted version of the messages on Facebook's servers in away that is transparent to Facebook functionality. Thus, users will continue to use Facebook as usual while maximizing the level of their privacy. The main issue with such approaches is that they are designed specifically for social networks and cannot be applied to other domains. Several researchers [13], [14], [16], [17] propose solutions to perform some sort of processing on encrypted data, such as search and information sharing. However, all these approaches involves modifying the database layer as well as the application code, which is not the focus of our work, where we try to maximize the data confidentiality without changing/breaking the functionality of the application. Trusted cloud computing platform (TCCP) has been proposed by [15]. The idea is to provide a closed-box execution environment for the consumers, guaranteeing that the cloud provider cannot tamper with the users data. Moreover, it allows the consumers to remotely check whether the server is running a TCCP implementation or not. The main limitation of this approach is the infrastructure provider, since they need to adopt TCCP first, then consumers are capable of using it. *CloudProtect* [19] and *Silverline* [20] are closely related to *HPISecure*. Both of them support keeping the data at the server-side confidential by encryption in away that is transparent to the application. However, *Silveline* proposed to dynamically analyze the application to determine which parts of the data can be functionally encryptable. It divides the users into groups, and assign a single encryption key to this group, facilitating encryption and information sharing at the same time. It assumes that any data is accessed by functions initiated by the user cannot be encrypted. In contrast to *Silverline*, *CloudProtect* encrypt all users' data and route users request to operate on it, if certain operations

requires data to be in plain text, it implements a protocol to expose this data for a short period of time. This would require an extra overhead, and so they introduced a relaxation policy to allow the user to trade-off security for performance.

VII. Conclusion

Data confidentiality is one of the key concerns in cloud computing. Organizations are not widely adopting the cloud because of issues related to security and privacy of their data. In this paper, we present *HPISecure*, a software prototype designed to facilitate the process of securing data stored on the cloud. *HPISecure* allows the user to store an encrypted version of his data on the cloud without breaking the functionality of the application. *HPISecure* intercepts the HTTP request/response objects, encrypt data before transmitting to the cloud, and decrypt data received back from the server. We have successfully tested *HPISecure* with Google Docs and Google Calender. There are still open challenges that we plan to cover in the future work, such as extending *HPISecure* to work with enterprise applications. Implement *HPISecure* as a web-based applications instead of desktop application. Trading off availability for security is a critical point that is not deeply covered by this work, we plan to include it in the next extension of *HPISecure*.

Acknowledgment

The authors would like to thank Mohammad AbuJarour from SAP for his valuable insights and feedback.

References

[1] A. Konary, S. Graham, and L. Seymour: *The future of software licensing: Software licensing under siege*. International Data Corporation, White Paper, 2004.

[2] Gartner website: *Software-as-a-Service Revenue*. [Online]. Available: http://www.gartner.com/newsroom/id/1963815 [retrieved: Mar, 2013]

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and M. Zaharia: *Above the Clouds: A Berkeley View of Cloud Computing*. Technical Report, University of California, Berkeley, USA, 2009.

[4] The AICPA website. Generally accepted privacy principles. [Online]. Available: http://www.aicpa.org/InterestAreas/InformationTechnology/Resources/Privacy/GenerallyAcceptedPrivacyPrinciples [retrieved: Jan, 2013]

[5] Cross-Tab Marketing Services. (Jan, 2010). *Online Reputation in a Connected World*. [Online]. Available: http://www.job-hunt.org/guides/DPD_Online-Reputation-Research_overview.pdf [retrieved: Jan, 2013]

[6] C. Wang, Q. Wang, K. Ren, and W. Lou: *Ensuring data storage security in cloud computing*. In Proc. IWQoS 09, Charleston, South Carolina, USA, 2009.

[7] A. Saha: *Computing on encrypted data*. In Proc. ICISS 2008, Hyderabad, India, 2008.

[8] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford, 2009.

[9] BoxCryptor website. [Online]. Available: http://www.boxcryptor.com [retrieved: April, 2013]

[10] TrueCrypt website. [Online]. Available: http://www.truecrypt.org [retrieved: April, 2013]

[11] Fiddler website. [Online]. Available: http://www.fiddler2.com [retrieved: April, 2013]

[12] M. M. Lucas and N. Borisov. *Flybynight: mitigating the privacy risks of social networking*. In Proc. of ACM WPES, Virginia, USA, 2008.

[13] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Strain. *Persona: An online social network with user-defined privacy*. In Proc. of ACM SIGCOMM, Barcelona, Spain, 2009.

[14] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. *Public key encryption with keyword search*. In proc. of Eurocrypt, Interlaken, Switzerland, 2004.

[15] N. Santos, K. P. Gummadi, and R. Rodrigues. *Towards trusted cloud computing*. In Proc. of HotCloud, San Diego, USA, 2009.

[16] H. Haclgümüs, B. Lyer, C. Li, and S. Mehrotra. *Executing SQL over encrypted data in the database-service-provider model*. In Proc. of ACM SIGMOD, Wisconsin, USA, 2002.

[17] H. Wang and L. V.S. Lakshmanan. *Efcient secure query evaluation over encrypted XML databases*. In Proc. of VLDB, Seoul, Korea, 2006.

[18] K. Zhang, Y. Shi, Q. Li, and J. Bian. *Data Privacy Preserving Mechanism based on Tenant Customization for SaaS*. In Proc. IEEE MINES, Wuhan, China, 2009.

[19] M. H. Diallo, B. Hore, E. C. Chang, S. Mehrotra, and N. Venkatasubramanian. *CloudProtect: Managing Data Privacy in Cloud Applications*. In Proc. IEEE Cloud, Hawaii, USA, 2012.

[20] K. P. N. Puttaswamy, C. Kruegel, and B. Y. Zhao. *Silverline: toward data confidentiality in storage-intensive cloud applications*. In Proc. of ACM SOCC, Cascais, Portugal, 2011.