Cloud Spreadsheets Supporting Data Processing in the Encrypted Domain

D. A. Rodríguez-Silva[1], L. Adkinson-Orellana[1], B. Pedrero-López[1] and F. J. González-Castaño[2]
[1]*Gradiant, Edif. CITEXVI, Campus de Vigo, 36310, Pontevedra, Spain*
[2]*AtlantTIC, Escuela de Ingeniería de Telecomunicación, Universidade de Vigo, 36310, Pontevedra, Spain*
{darguez, ladkinson}@gradiant.org, javier@det.uvigo.es

Keywords: Cloud Computing, Security, Privacy, Homomorphic Encryption, Spreadsheet.

Abstract: Security has become one of the main barriers for the adoption of cloud services. A range of legal initiatives that require support mechanisms such as access control and data encryption have been proposed to ensure privacy for data moved to the cloud. Although these mechanisms are currently feasible in situations in which the cloud acts as a mere data storage system, they are insufficient in more complex scenarios requiring processing in external cloud servers. Several new schemes have been proposed to overcome these shortcomings. Data Processing in the Encrypted Domain (DPED) permits arithmetic operations over ciphered data and the generation of encrypted results, without exposure of clear data. In such a set-up, the servers have no access to the information at any point of the process. In this paper we describe, as a case study of secure cloud data processing, a cloud spreadsheet that relies on DPED libraries to perform operations in the encrypted domain. Tests performed on local servers and in the Google cloud through the Google App Engine platform show that representative real applications can benefit from this technology. Because the proposed solution is PaaS-oriented, developers can apply the libraries to other applications.

1 INTRODUCTION

Security and privacy are both major concerns for Cloud Computing users. As reported in the European CIOs and Cloud Services research study (2010), around 71% of European companies are worried about security and privacy, especially when it comes to storing or processing sensitive data in the cloud. Security has thus become a significant barrier to full adoption of cloud services.

Concerns regarding security and privacy have been addressed in part by different legal initiatives within the European Union, such as Directive 95/46/EC of the European Parliament and the Council of October 24 1995 (Data Protection Directive, 1995), which proposes a set of recommendations for protecting personal data during transfer and processing. In Spain there are several specific laws to protect and regulate the management of personal and corporate data used by cloud applications, including the Data Protection Regulation, of Law 15/1999 on Personal Data Protection (LOPD, 1999) and the Royal Decree 1720/2007, which approves the development of the LOPD (RDLOPD, 2007). As an example of the recommended proposals, the 85th article of the RDLOPD states that security measures applied to personal data in communication networks, public or not, should guarantee at least the same security level as that offered by local access systems.

Due to their very nature, data processed in the cloud will presumably be affected by international data transfers, primarily because many web applications are hosted on foreign servers. International data movement is regulated by the data protection regulation, which forbids international data transfers between countries that do not offer sufficient security guarantees according to the LOPD, although there are some exceptions explicitly indicated in the reference regulation. In addition, this regulation sets out several legal requirements, such as transfer notification to the Spanish Data Protection Agency.

Because legal procedures are slow, new technological mechanisms are required until the situation is completely regulated. Authentication on the client side and use of security mechanisms such as data encryption during data transmission are good solutions for interception attacks and servers that do not offer sufficient guarantees of reliability. To increase security, the client can cipher data using a

private key, thereby hiding the information from the server. Although this option is valid when the cloud acts as a mere storage service, there are cases in which it would be insufficient, for example when performing certain calculations on the server or when processing a query to a database with ciphered data.

To overcome the above shortcomings, new server-side schemes have been proposed, such as the use of cryptographic hardware or Data Processing in the Encrypted Domain (DPED). Cryptographic hardware can be used to perform cryptographic operations and to store keys securely, but it is expensive (specific trusted anti-tampering devices are required) and it needs to be physically integrated into the provider's infrastructure. DPED overcomes these problems, but at the expense of increased processing time. It enables operations over ciphered data that generate encrypted results, thereby allowing server-side operations without revealing the original information. This adds an additional security level to the cloud paradigm by means of complex homomorphic algorithms. The computational requirements may not be a problem thanks to the scalability and flexibility of the cloud paradigm.

In this paper we describe a cloud spreadsheet application that uses the DPED concept to perform operations in the encrypted domain. We have tested it on our local servers and in the Google cloud through the Google App Engine (GAE) platform. Section 2 discusses related work and section 3 explains the implementation details of our application. Section 4 presents the tests performed, and finally, section 5 concludes the paper.

2 RELATED WORK

Many office cloud applications allow users to work with spreadsheets. Some well-known examples are Microsoft Office 365, Google Drive Spreadsheets, Thinkfree Calc and Zoho Sheet. Nevertheless, none of these applications currently offers full protection mechanisms for user data, meaning that privacy, when available, is supported by external means. Indeed, most current solutions are designed for Google Drive, not for spreadsheets. Furthermore, although there are solutions that are completely integrated with the Google Drive interface that encrypt documents transparently to users (Adkinson-Orellana et al., 2010), most simply use the cloud to store the encrypted documents (CryptRoll, 2013; ZecurePC, 2011; and CloudLock, 2015).

DPED allows certain operations to be performed over ciphered data without the need to access the clear version. In particular, arithmetic operations can be performed efficiently in the encrypted domain thanks to the concept of additive and multiplicative privacy homomorphisms (Brickell and Yacobi, 1987). In 2009, Gentry presented the first fully homomorphic encryption scheme. He described public key encryption using ideal lattices (Gentry, 2009). In the same year, M. Van Dijk described a "somewhat homomorphic" encryption scheme based on elementary modular arithmetic, and used Gentry's techniques to convert it to a full homomorphic scheme (M. Van Dijk et al., 2009) that implemented addition and multiplication over integers rather than ideal lattices over a polynomial ring.

There have been other contributions in this direction. A. F. Chan formulated a privacy homomorphism for operating over ciphered data with two different encryption schemes, where data could be processed directly in an encrypted form (Chan, 2009). H. Hacigümüş, in turn, described different techniques for executing SQL queries over encrypted data (Hacigümüş et al., 2002). The strategy involves processing as much of the query as possible at the service provider site, without decrypting data. Decryption and the remainder of the query processing takes place at the client side. They also explored an algebraic framework to split the query to minimize computation at the client side.

The innovative idea in this paper is to enable DPED processing in cloud applications. We are not aware of any previous DPED-enabled complex cloud applications, although, in a previous work, we presented a toy example that demonstrated that DPED could strengthen the privacy of simple mathematical operations in the cloud (Rodriguez-Silva et al., 2011).

3 SECURE CLOUD SPREADSHEET

3.1 Arithmetic Calculations in the Encrypted Domain

The spreadsheet application is composed of two modules: a client module and a server module. The client module presents the spreadsheet interface, which is used to enter data, cipher its content, send it to the server, and decipher and present the results in the corresponding spreadsheet cell. The server

module, in turn, executes arithmetic operations on the encrypted data received from the client by means of adequate privacy homomorphisms. The supported encrypted operations are listed in Table 1.

Due to the low efficiency of complete homomorphisms in the current state-of-the-art, our implementation uses a variation of the additive homomorphic encryption described by Paillier (Paillier, 1999) as the basis of our cryptographic system. One or more additional rounds of communication between the client and the server will also be needed depending on the complexity of the operation requested.

Table 1: Encrypted operations supported by the spreadsheet.

Operation	*Description*	*Example*
AVERAGE	Average value	AVERAGE (A1:A5)
DEGREES	Degree conversion	DEGREES (A1:A5)
FFT	Fast Fourier Transform	FFT (A1:A5)
PROD	Product	PROD (A1:A5)
RADIANS	Radian conversion	RADIANS (A1:A5)
SPROD	Scalar product	SPROD(A1:A5;B1:B5)
STDEV	Standard deviation	STDEV (A1:A5)
SUM	Addition	SUM (A1:A5)
VADD	Vector addition	VADD (A1:A5;B1:B5)
VAR	Variance	VAR (A1:A5)
VPROD	Vector product	VPROD(A1:A5;B1:B5)
VSUB	Vector substraction	VSUB (A1:A5;B1:B5)

The encryption methods used are based on asymmetric key algorithms. The libraries present different options, such as threads and JNI (Java Native Interface), thereby increasing efficiency thanks to the use of C libraries. The encryption libraries also allow operations with scalars and vectors, unlike the version in our previous work, which only offered basic operations for unary values.

3.2 Ciphered Cloud Spreadsheet Implementation

The spreadsheet allows operations over a range of cells, with no limitations in terms of the number of operators involved. The implementation relies on Java technology, as this is the most common PaaS language. It uses the Java Runtime Environment (JRE) classes available to create applets and tables (JTable, TableModel, etc.), meaning that results can be easily embedded on a web page. The development environment used to create the spreadsheet and the associated technologies are the same as those used for the encrypted calculator (Rodriguez-Silva et al., 2011): Java Servlets and IDE Eclipse 3.5 (Galileo) for the server and Java applets and Oracle IDE Netbeans 6.8 for the client, with the corresponding plugin to create graphical user interfaces. Again, GAE was selected as the cloud platform to deploy the application. Due to the restrictions of this PaaS, the encryption libraries had to be adapted, since the platform has a limited support for multithreading (a characteristic that the libraries use to improve efficiency).

By default, some of the applet functionalities (e.g. reading or writing files on disk) are restricted through a security policy implemented by the security controller of the browser Java Virtual

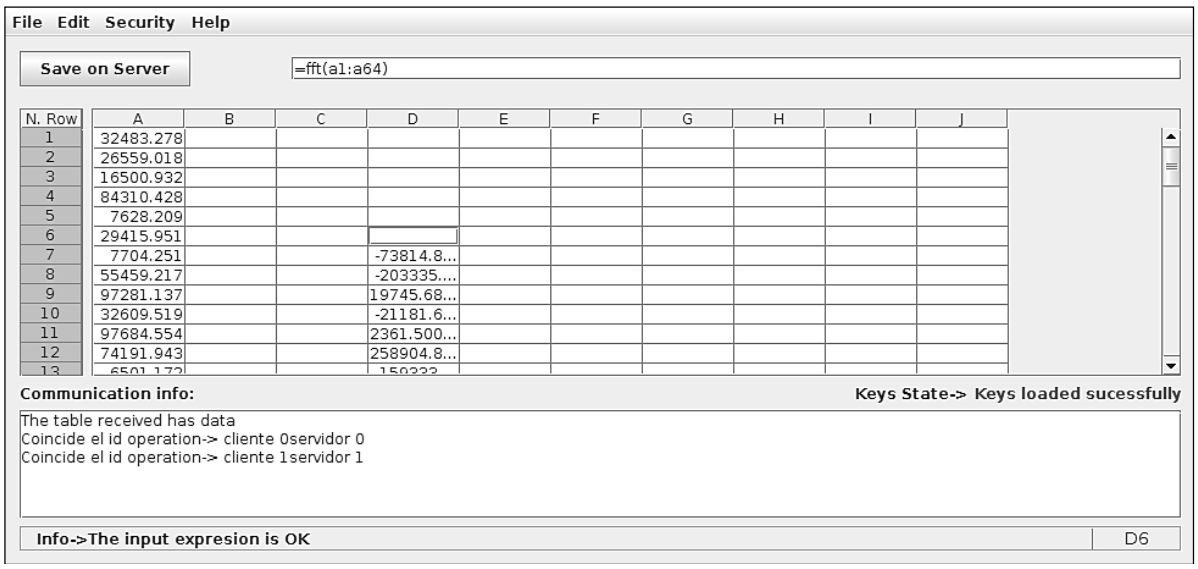

Figure 1: Ciphered cloud spreadsheet interface.

Machine (JVM) plugin. However, the spreadsheet applet needs these functionalities to store its content in a local file and to obtain and save the user keys to cipher data. For this reason, the user must accept the applet signature, granting certain restricted functionalities. All the libraries except `au.com.bytecode.opencsv` (required to save the content of the spreadsheet in .CSV format) and the DPED libraries are available on the JRE used by the browsers. Other libraries are downloaded when the applet starts.

The client is composed of two modules: one for the graphic interface and the other for parsing and communicating with the server. The graphic module is in charge of the visual interface, intercepting user events and presenting results. Its design was based on typical spreadsheet software, such as Google Spreadsheet, OpenOffice.org and Microsoft Excel (Figure 1). The client parser analyses formulae expressions to obtain the data required to perform the operations. Finally, the communication module exchanges data with the server using HTTP tunnelling.

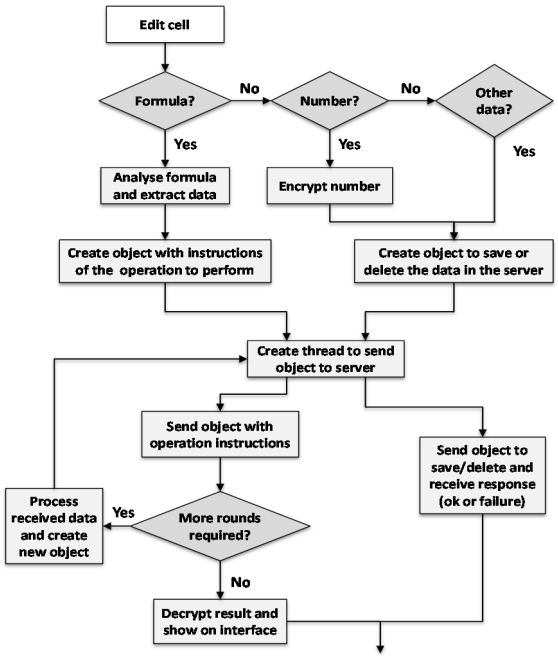

Figure 2: Client module flowchart according to user interaction.

To perform a spreadsheet operation, the first step is to start the application (applet) and load the libraries. The user then introduces his/her login and password, using his/her Google user account if the application is deployed in GAE. If access is granted, the user keys —required to cipher and decipher the data and perform encrypted operations— are loaded from binary files. When the process is complete, the spreadsheet graphic interface is shown. At this point, the application is ready to process events generated by user interaction (see Figure 2):

- *Finish cell edition.* The application checks the type of data entered in the cell, such as formulae, a number or other data types. The formulae are analysed, the type of operation and the references to the cells are extracted, and in the case of numbers, these are ciphered. With this information a new communication object is created to be sent to the server. In the case of other data types, the cell content is directly inserted in the communication object. Once this object has been created, a new communication thread is thrown to send the object to the server. In other words, the interface thread is released to receive new user events. The communication thread remains open until confirmation is received that the number or object has been correctly received at the server side or until the operation defined by the formula has finished. This is indicated by the communication round. The result is then decrypted and displayed to the client.
- *Save on server.* The client creates an object with the order to save the spreadsheet at the server side. A new thread is created specifically to send the petition containing this object. This thread will receive a response indicating the success or failure of the request.
- *Select a menu option.* The selected option is performed at the client or the server side, depending on the actions involved, e.g., create/load user keys, add/remove rows or columns, copy/paste cells, etc.

Dependencies between the values of the cells must be taken into account. When the value of a cell changes, it can affect other related cells, resulting in the execution of multiple parallel operations. To manage this situation, a new thread is thrown for each dependent operation, creating a new client. This ensures that the different operations executed do not interfere with each other, but it requires more processing load for the application during the initialization of a new module.

We considered two possible server deployment scenarios: a private cloud (local) and a public cloud (GAE). The private cloud does not have all the resources and services offered by GAE, such as user accounts and persistent storage. In this case users are authenticated through a local mechanism and persistent storage is simulated by saving the spreadsheet in a local file, identified by the user

login. In this way, when a user enters the application, the saved spreadsheet will be retrieved and deciphered if and only if he/she has the appropriate private key.

The server is a servlet composed of two modules: the communication module and the data processing module. The communication module performs the same actions as on the client side, while the data processing module is in charge of storing or processing the data received in the encrypted domain and recovering the stored spreadsheet.

On the server side, the first step is to initialize the servlet, which will keep listening to incoming petitions from the clients. When a request is received, the thread recovers the object and retrieves the data it contains. The selected action is then executed, i.e. the data received is saved, the cell value is deleted, the module is initialized, the encrypted operation is performed, etc.

The client and server modules must store the status of each operation requested, indicating the current execution round. If several operations are requested at once, several modules for the client and the server will be instantiated, and those associated with the same operation will be identified by a unique identifier. Thus, each time an operation is requested, it will be possible to execute it in the corresponding module, avoiding result inconsistencies when several rounds are being carried out.

Communication between the client and the server takes place through a Java object, which is used to retrieve and store information. This object is sent through HTTP tunnelling, facilitating data transmission through different elements (firewalls, proxies, etc.) that typically limit connection to web resources. In addition, the object is used to send and receive different types of data, such as encrypted data to be stored by the server, information related to the operation performed and the cells involved, encrypted results received from the server, etc.

The application also supports the generation of the keys required to cipher and decipher data. These keys offer two levels of security: short-term and medium-term. While short-term security speeds up encryption and decryption, medium-term security is stronger, as it would take approximately ten times longer to break up its keys (e.g., ten years vs 1 year).

4 PERFORMANCE TEST

The test layout comprised a client computer (Intel Core i3-2120 @ 3.3 GHz, 3870 MB RAM, Ubuntu 10.4) and a local server (with the same characteristics) to perform the encrypted operations under Jetty 7.5.4. The cloud application applet was executed through the Google Chrome 17 browser. We also used GAE servers equipped with Jetty to deploy the applications.

The operation selected to evaluate the performance of the spreadsheet was a Fast Fourier Transform (FFT), as this is a complex operation that permits representative performance results. The FFT was applied to vectors with lengths of 64, 128 and 512 points.

The current version of the spreadsheet encrypts and sends each data item to the server individually. When an operation (the FFT in this case) is selected at the client side, the server simply receives the operation and the cells involved, since it has already the ciphered values. Therefore, the total time needed to perform an operation comprises two times: a *data entry time*, including the management of the data in the cells in the spreadsheet and the time needed to encrypt and send each operand to the server, and a *running time*, which is the time needed to perform an encrypted operation on the server and present the result on the client side.

We used four test scenarios:

- *Local, with threads and JNI.* The encrypted FFT was executed on the local server. There were five parallel threads in the server and the client. JNI was used to improve efficiency.
- *Local, without threads or JNI.* As in the previous case, the operation was executed on the local server. Neither threads for parallel executions nor C functions were used in this scenario.
- *Local, clear FFT.* We executed the FFT using clear data on the local server. The FFT algorithm was implemented by a Java function.
- *Remote, deployed on a GAE server*. GAE does not allow the use of threads or JNI, so the server was subject to these restrictions. At the client side five threads were used, in addition to C functions through JNI.

The client was executed on the same machine and used the same browser in all four scenarios.

In each scenario, 10 FFT operations were performed for randomly generated vectors for the three lengths. We used both the short- and medium-term security levels to perform these tests. Tables 2 and 3 show the corresponding results based on the following times:

- *Entry time (ET).* Time from the moment an

operand is entered in the spreadsheet to the moment the client receives the response from the server (indicating that the encrypted data have been correctly stored).

- *Server execution time (SE).* Time taken by the server to perform the encrypted FFT operation.
- *Running time (RT).* The sum of communication time required to exchange data between the client and the server, the time needed by the server to perform an ciphered FFT (SE) and the time needed to decrypt the result for the client.

Each pair of values represents the average execution time ($\bar{x}$) and its standard deviation (σ), both in milliseconds.

The tests were unfeasible for the 512-point FFT in the remote scenario because the time needed to generate and return a response in GAE is limited to 30-60 seconds and the 512-point FFT needs longer. These results are therefore not shown in the tables.

The use of longer keys improves security considerably, but increases effective operating time. On comparing Table 2 and Table 3, we can see that the time required to perform the same operation is considerably higher for the medium-term security level. This mainly affects ciphered operation time (i.e., server execution time). The different level of security does not have an impact on data entry time, as this includes the ciphering of data but not the execution of the encrypted operations.

The use of threads and JNI considerably reduced the execution time in both cases, primarily due to the improved efficiency of the execution of the algorithm on the server and the improved efficiency of the decoding process. Although this test scenario cannot be translated to GAE, its results can give us an idea of how the performance could be improved with JNI and threads in GAE or other compatible PaaS.

The running time with GAE was much higher than in the equivalent local case without threads or JNI. Besides of the Internet delay, GAE servers took approximately 10 times longer to execute the same algorithm, probably because GAE is optimized for applications with short response times, typically of hundreds of milliseconds.

The total execution time for the best DPED scenario (local, threads and JNI) was much longer than with unencrypted data (local, clear FFT). This is obviously due to the time spent on encrypting data and decrypting the result, and the efficiency of the DPED FFT algorithm, which is over 200 times slower than the algorithm used to calculate the FFT with unencrypted data.

Table 2: Data entry time, server execution time and running time using the short-term security level (ms).

Test scenario		***64***	***128***	***512***
Local, no threads or JNI	ET	$\bar{x}$=524.2 σ=23.5	$\bar{x}$=897.9 σ =43.3	$\bar{x}$=6888.4 σ =114.4
	SE	$\bar{x}$=581.4 σ=152.7	$\bar{x}$=1177.9 σ =137.8	$\bar{x}$=6013.7 σ =120.1
	RT	$\bar{x}$=1206.6 σ=140.4	$\bar{x}$=2256.7 σ =135.1	$\bar{x}$=9918.4 σ =131.5
Local, threads and JNI	ET	$\bar{x}$=565.1 σ =21.9	$\bar{x}$=936.1 σ =21.3	$\bar{x}$=8642.9 σ =127.6
	SE	$\bar{x}$=76.9 σ=9.4	$\bar{x}$=140.9 σ =31.7	$\bar{x}$=545.8 σ =62.5
	RT	$\bar{x}$=239.1 σ =18.3	$\bar{x}$=356.3 σ =41.4	$\bar{x}$=1108.8 σ =103.6
Remote (GAE)	ET	$\bar{x}$=16539.3 σ =538.9	$\bar{x}$=34056.3 σ =1355.2	---
	SE	$\bar{x}$=5595.2 σ=294.3	$\bar{x}$=13793.8 σ =858.8	---
	RT	$\bar{x}$=6339.3 σ =365.7	$\bar{x}$=14688.0 σ =881.2	---
Local, clear FFT	ET	$\bar{x}$=475.6 σ =23.9	$\bar{x}$=790.3 σ =41.3	$\bar{x}$=6250.7 σ =118.3
	SE	$\bar{x}$=0.159 σ=0.086	$\bar{x}$=0.516 σ =0.307	$\bar{x}$=1.318 σ =0.422
	RT	$\bar{x}$=6,5 σ =1.5	$\bar{x}$=6.9 σ =2.5	$\bar{x}$=10.4 σ =2.0

Table 3: Data entry time, server execution time and running time using medium-term security level (ms).

Test scenario		***64***	***128***	***512***
Local, no threads or JNI	ET	$\bar{x}$=1132.3 σ=35.1	$\bar{x}$=1916.8 σ=36.9	$\bar{x}$=8308.8 σ=192.5
	SE	$\bar{x}$=2382.4 σ=500.8	$\bar{x}$=4538.1 σ=623.2	$\bar{x}$=20909.2 σ=742.1
	RT	$\bar{x}$=6127.7 σ=550.3	$\bar{x}$=11129.8 σ=567.9	$\bar{x}$=46873.3 σ=1314.3
Local, threads and JNI	ET	$\bar{x}$=589.9 σ=53.6	$\bar{x}$=995.1 σ=38.3	$\bar{x}$=6983.7 σ=137.4
	SE	$\bar{x}$=173.6 σ=23.2	$\bar{x}$=337.0 σ=35.7	$\bar{x}$=1558.2 σ=287.5
	RT	$\bar{x}$=597.4 σ=46.6	$\bar{x}$=1031.6 σ=35.1	$\bar{x}$=4062.7 σ=491.0
Remote (GAE)	ET	$\bar{x}$=16733.6 σ=1113.3	$\bar{x}$=34709.2 σ=2410.8	---
	SE	$\bar{x}$=22045.1 σ=915.4	$\bar{x}$=52640.6 σ=1033.4	---
	RT	$\bar{x}$=23007.0 σ=926.6	$\bar{x}$=54172.3 σ=1111.0	---
Local, clear FFT	ET	$\bar{x}$=475.6 σ =23.9	$\bar{x}$=790.3 σ =41.3	$\bar{x}$=6250.7 σ =118.3
	SE	$\bar{x}$=0.159 σ=0.086	$\bar{x}$=0.516 σ =0.307	$\bar{x}$=1.318 σ =0.422
	RT	$\bar{x}$=6,5 σ =1.5	$\bar{x}$=6.9 σ =2.5	$\bar{x}$=10.4 σ =2.0

5 CONCLUSIONS

Cloud computing provides an adequate environment for deploying applications following a Software-as-a-Service (SaaS) model. However, security and privacy are key concerns when sensitive data managed by applications is moved to cloud infrastructures for processing or storage.

In this paper we have proposed, as a case study of a real-life secure cloud application, a spreadsheet capable of performing DPED operations on cloud servers. The application was tested on a private cloud and on GAE, with analysis of the time required to perform a ciphered FFT operation. Although the test results demonstrate that homomorphic encryption is a feasible solution for secure data processing on cloud infrastructures, the efficiency of current encrypted domain libraries needs to be improved to achieve commercial status. Nevertheless, although the times for encrypted operations are quite long, they are satisfactory for applications with a light processing load, such as the proposed spreadsheet. To apply this model in a PaaS, cloud providers should support DPED libraries on their servers.

This solution could be applied to other real-life applications, such as enterprise resource planning (ERP) or e-Health SaaS, where confidentiality is crucial.

ACKNOWLEDGEMENTS

This research was supported by the SAFECLOUD grant (09TIC014CT), funded by Xunta de Galicia (Spain), and partially supported by the HIGEA grant (IPT-2012-1218-300000), funded by the Spanish Ministry of Economy and Competitiveness, the PRISMED grant (IPT-2011-1076-900000), funded by the Spanish Ministry of Science and Innovation. This research was conducted with the collaboration of GPSC research group of the University of Vigo, which provided the DPED libraries, and Fundación Barrié.

REFERENCES

Adkinson-Orellana, L., Rodríguez-Silva, D. A., Gil-Castiñeira, F., and Burguillo-Rial, J., 2010. Privacy for Google Docs: Implementing aTransparent Encryption Layer. In *Proc. of 2nd Cloud Computing International Conference–CloudViews 2010 (pp. 20-21).*

Brickell, E. F., Yacobi, Y., 1987. On Privacy Homomorphisms. In *Advances in Cryptology–EUROCRYPT 87* (pp. 117-125). Springer Berlin Heidelberg.

Chan, A. F., 2009. Symmetric-key homomorphic encryption for encrypted data processing. In *Communications, 2009. ICC'09. IEEE International Conference on* (pp. 1-5). IEEE.

CloudLock. [Online]. [Accessed 6 January 2015]. Available from: http://www.cloudlock.com/

CryptRoll.2013. [Online]. [Accessed 6 January 2015]. Available from: http://cryptroll.android.informer.com/

Data Protection Directive. [Online]. [Accessed 6 January 2015]. Available from: http://ec.europa.eu/justice/data.protection/index_en.html.

European CIOs and Cloud Services, 2010. [Online]. [Accessed 6 January 2015]. Available from: http://www.colt.net/cio-research.

Gentry, C., 2009.Fully Homomorphic Encryption Using Ideal Lattices. In*41st ACM Symposium on Theory of Computing–STOC* (Vol. 9, pp. 169-178).

Hacigümüş, H., Iyer, B., Li, C., and Mehrotra, S., 2002. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data* (pp. 216-227). ACM.

LOPD, Ley orgánica 15/1999 de Protección de Datos de Carácter Personal, Boletín Oficial del Estado (in Spanish), 1999.[Online].[Accessed 6 January 2015]. Available from: https://www.boe.es/

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology–EUROCRYPT'99* (pp. 223-238).Springer Berlin Heidelberg.

RDLOPD, Real Decreto 1720/2007, Reglamento de Desarrollo de la LOPD, BoletínOficialdel Estado (in Spanish), 2007.[Online]. [Accessed 6 January 2015]. Availablefrom: https://www.boe.es/

Rodriguez-Silva, D. A., González-Castaño, F. J., Adkinson-Orellana, L., Fernández-Cordeiro, A., Troncoso-Pastoriza, J. R., and González-Martínez, D., 2011. Encrypted Domain Processing for Cloud Privacy. Concept and Practical Experience. In *Proceedings of 1st International Conference on Cloud Computing and Services Science–CLOSER 2011.*

Van Dijk, M., Gentry, C., Halevi, S., and Vaikuntanathan, V., 2010. Fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2010* (pp. 24-43). Springer Berlin Heidelberg.

ZecurePC. 2011. [Online]. [Accessed 6 January 2015]. Available from: http://www.zecurex.com/.