# Leveraging an Homomorphic Encryption Library to Implement a Coordination Service

Eugenio A. Silva Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

*Abstract*— **The paper presents MorphicLib, a new partial homomorphic cryptography library written in Java that can be used to implement a wide-range of applications. The paper shows the use of the library with the HomomorphicSpace coordination service. This service is a tuple space that stores encrypted tuples but still supports operations like returning tuples with values within a certain range.**

## I. Introduction

Researchers and practitioners for decades assume that encrypted data cannot be processed. Generally, it is necessary to decrypt the data before performing any operations over that data. This problem becomes specially important when large data resides in a public cloud. In this case, there is a dilemma between two alternatives: (1) either the data is decrypted in the server-side (in the cloud), which poses security issues, namely the need to pass the key to the server and have the information exposed to insider threats in the cloud [1], [2] at least during the operation; or (2) the data is decrypted in the client-side, which involves downloading the data from the cloud (typically expensive and slow) and prevents using the computation power of the cloud. A good solution to this dilemma would be to perform the desired operations directly on the encrypted data, at the server-side, where it is stored. This would avoid the cost of moving the data, would allow leveraging the cloud's computation power, and would solve the security issues.

The term *homomorphic encryption* designates forms of encryption that allow some operations to be performed over encrypted data, without decrypting it. With those forms of encryption, it is possible to perform resource-intensive computing tasks at server-side without having to decrypt it first. Homomorphic encryption became popular with Gentry's work [3], which was coincident with the emergence of cloud computing. Gentry's scheme provides fully homomorphic encryption (FHE), so it allows performing arbitrary computation on encrypted data. Other FHE schemes were presented in the following years [4], [5].

Although in theory FHE solves the problem of computing encrypted data outsourced to a cloud, the performance of these schemes is too poor for practical applications [6]. For that reason, much effort has been placed in developing and using *partial homomorphic encryption* (PHE) schemes [7], [8], [9], [10], [11]. PHE schemes allow performing some computation over encryption data, but not arbitrary computation like FHE. CryptDB is an important step towards the deployment of PHE in real systems [11]. CryptDB is a relational database management system that stores encrypted data and allows doing SQL queries. The system combines a set of PHE schemes and has enough performance for many applications.

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

This paper presents *MorphicLib*, a new partial homomorphic cryptography library that can be used to implement a wide-range of applications. The library contains functions (normally) executed at the client-side and functions for the server-side. For the client-side there is the encryption scheme, i.e., functions for encryption, decryption, and key generation. For the server-side there are homomorphic equivalent operations (addition, multiplication, comparison, etc.). The library was programmed in Java in order to ensure portability, i.e., that it can be executed in different platforms, both client and server-side. Moreover, Java is arguably the most popular general purpose programming language today, with a large set of APIs, and a strong programming community.

The paper shows the usefulness of the library with a service that is interesting in its own right, the *HomomorphicSpace coordination service*. Coordination services like Google Chubby [12], Google Megastore [13], Apache Zookeeper [14], and GigaSpace's tuple space (now part of XAP) [15] are important components in current cloud systems. They are used for tasks such as synchronization, locking, orchestration, metadata storage, leader election, and replica failure detection. *DepSpace* [16], [17] is a tuple space, i.e., a coordination service that follows Linda's associative memory paradigm [18], similarly to GigaSpace's tuple space. DepSpace is replicated, so it can tolerate arbitrary (Byzantine) faults in some of its replicas.

HomomorphicSpace is an extension of DepSpace with homomorphic encryption (MorphicLib), so that data (tuples) can be stored encrypted at the servers. DepSpace's commands to read and retrieve tuples were extended with operators for inequality, less/greater relations, and keyword search, all over encrypted data. Moreover, HomomorphicSpace supports addition and multiplication of tuples in the server. Data is never decrypted at the server, only at the client after retrieval. HomomorphicSpace is Byzantine fault-tolerant like DepSpace.

## II. MorphicLib Library

As already mentioned, MorphicLib is a novel library of partial homomorphic cryptographic functions written in Java and providing a Java API. MorphicLib was not developed from scratch, but based on existing source code whenever possible. The objective was both to simplify the task and to avoid introducing bugs, which tend to appear due to the complexity of cryptographic code. This library can be used both at the

TABLE I
MORPHICLIB'S MAIN CLASSES

| **Property** | **Homomorphic Operations** | **Class** | **Input Data Types** |
|---|---|---|---|
| Random | None (strong cryptanalisys resistance) | `HomoRand` | Strings, Byte Arrays |
| Deterministic | Equality an inequality comparisons | `HomoDet` | Strings, Byte Arrays |
| Searchable | Keyword search in text | `HomoSearch` | Strings |
| Order preserving | Less, greater, equality comparisons | `HomoOpeInt` | 32 bit Integers |
| Sum | Add encrypted values | `HomoAdd` | BigInteger, String |
| Multiplication | Multiply encrypted values | `HomoMult` | BigInteger, String |

client-side to encrypt and decrypt data, and at the server-side to do operations over encrypted data.

The code of the library is organized in classes, one per *homomorphic property*. One crucial different between PHE and FHE is that in the former data has to be encrypted taking into account the kind of operation that will be supported over the encrypted data. With FHE, on the contrary, arbitrary computation is possible over encrypted data (at a cost, in terms of performance). As we opted for PHE, for each homomorphic operation we have four kinds of functions (or methods):

- key generation function, typically used at client-side;
- encryption function, typically used at client-side;
- decryption function, typically used at client-side;
- homomorphic operation functions, which do operations over encrypted data, typically used at the server-side.

Information about the properties of the PHE algorithm, the operations supported, and the classes is in Table I.

## III. HOMOMORPHICSPACE COORDINATION SERVICE

This section presents HomomorphicSpace, a coordination service that leverages MorphicLib to handle encrypted data at the server. HomomorphicSpace is an extension of DepSpace, so we start by presenting the latter.

### A. DepSpace

DepSpace (Dependable Tuple Space) is a fault- and intrusion-tolerant *tuple space* [16]. Architecturally it is client-server system implemented in Java. The server-side is replicated in order to tolerate arbitrary faults. The client-side is a library that can be called by applications that use the service. Clients communicate with the servers using a Byzantine fault-tolerant total order broadcast protocol called BFT-Smart. The most recent version supports extensions to the service [17].

The service provides the abstraction of tuple spaces. A tuple space can be understood as a shared memory that stores *tuples*, i.e., sequences of *fields* (data items) such as (1, 2, a, hi). Tuples are accessed using *templates*. Templates are special tuples in which some fields have values and others have undefined values, e.g., wildcards meaning any value ("*"). A template *matches* any tuple of the space that has the same number of fields, in which the values in the same position are identical, and the undefined values match in some sense. For example, the template (1, *, a, *), matches the tuples (1, 2, a, hi) and (1, 7, a, 14), but neither (1, 2, b, 4) nor (1, 2, a, hi, 5). DepSpace supports a set of commands, issued by clients and executed by the servers. Here we consider the following commands:

- `out` *tuple* – inserts a tuple in the space;
- `inp` *template* – reads and removes from the space a tuple that matches the template;
- `rdp` *template* – reads but does not remove from the space a tuple that matches the template;
- `inAll` *template* – reads and removes from the space all tuples that match the template;
- `rdAll` *template* – reads but does not remove from the space all tuples that match the template.

DepSpace does not support homomorphic operations. However, it allows fields to be encrypted and basic equality matching by storing a hash jointly with the encrypted field. This solution however is vulnerable to trivial brute force and dictionary attacks. It does support the definition of access control policies using its policy-enforcement mechanism.

### B. Threat Model

The threat model we consider for HomomorphicSpace is similar to the threat model for DepSpace except for one crucial difference: we consider that any server may be adversarial and try to read the content of the tuples it stores. We consider that all tuples of their fields for which confidentiality has to be preserved are encrypted using homomorphic encryption, preventing malicious servers from doing such an attack. Similarly to DepSpace, adversaries may compromise up to $f$ out of $3f + 1$ servers and stop them or modify their behavior arbitrarily. This is tolerated using replication and the BFT-Smart protocol. Network messages may also be tampered with by the adversary, but the system uses this using secure channels.

### C. Commands

HomomorphicSpace extends DepSpace to allow commands over tuples with encrypted data items. More precisely in comparison with DepSpace, HomomorphicSpace: (1) supports the original match operations over encrypted data; (2) extend matching beyond the equality and wildcards with more complex matches, i.e., inequality, order comparisons (lower, greater), and keyword presence in a text, all over encrypted data; (3) allow addition and multiplication of encrypted fields.

Besides values and wildcards ("*"), HomomorphicSpace's *templates* can include the following fields:

- % $word_1 \ldots word_n$ – matches a textual field containing all the words indicated;
- $>$ *val* – matches a numeric field containing a value greater than *val*;
- $>=$ *val* – matches a numeric field containing a value greater or equal to *val*;
- $<$ *val* – matches a numeric field containing a value lower than *val*;
- $<=$ *val* – matches a numeric field containing a value lower or equal to *val*.

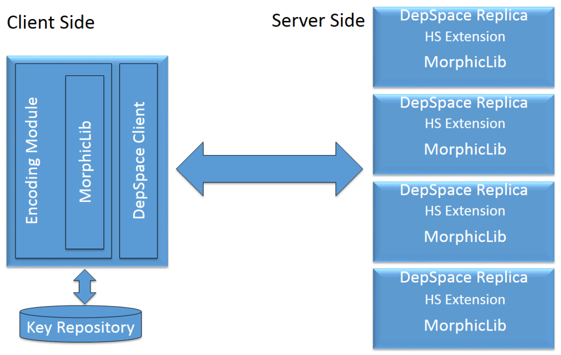

Fig. 1. HomomorphicSpace architecture

HomomorphicSpace adds three commands to those provided by DepSpace (Section III-A). The first is `crypt` *id template* and aims to define a *tuple encryption type*. The command takes as input an identifier (id) for the type it will create, and a template with the homomorphic operation desired for each of the fields, which will determine the homomorphic property. For example, if the template contains for a given field the operation "=", the system infers that the encryption to be used for that field is deterministic, which is the strongest that allows that operation. If no operation is indicated, the field will not be encrypted. The complete list of interpreted operations is:

- $=, <>$ – determinist encryption
- $>, >=, <, <=$ – order preserving encryption
- % – searchable encryption
- $+$ – Paillier
- & – RSA
- . – random encryption
- other value – no encryption

The second command is `rdSum` *template*. This command starts by collecting all the tuples that match the template similarly to `rdAll`, then sums the (encrypted) fields with $+$ in the template. The function returns a single tuple with the result. The third command is `rdProd` *template*, which works similarly to `rdSum` but does multiplication instead of sum.

This scheme allows a single type of encryption per field (unlike CryptDB). However, with the tuple data structure this is not a restriction. For instance, for tuples with a single numeric field, two operations like equality and sum can be supported by transforming that field in two and using the tuple encryption type $(=, +)$.

### *D. Architecture and Functioning*

Architecturally the HomomorphicSpace is similar to DepSpace, with a client-side and a server-side. Figure 1 represents the system with 4 replicas, i.e., with $f = 1$. From the confidentiality point of view, the server-side is untrusted and the client-side trusted.

The server-side of the system is mostly DepSpace code with the server-side of the MorphicLib and with extensions to process the homomorphic operations. The client-side includes MorphicLib's and DepSpace's client-side libraries. The main functions of the client is to encrypt tuples and send them to the tuple space, and to decrypt them before they are delivered to the application. When a tuple is encrypted, the encryption keys are stored in a *key repository* (a folder with one file per key). Next we describe both sides in more detail.

*Client side* – When the `crypt` command is issued (i.e., that method is called), the library generates keys for every field of the tuple for which homomorphic properties are desired. These keys are stored jointly with the tuple encryption type (id and template) in the key repository. All the other commands (`out`, `inp`, etc.) include an id that the library uses to retrieve the corresponding tuple encryption type and keys from the repository. If the operation indicated in a field is not compatible with the encryption defined with the `crypt` command, the command returns an error.

The library uses the DepSpace client library to send to the servers the command and the fields. If the command is an `out`, the fields are encrypted with the scheme defined in the tuple encryption type and the keys previously stored. If the command involves reading tuples, it contains the operation and encrypted values. Note that each field of each id has its own key (or key pair for RSA), but the same field for the same id is always encrypted with the same key. When the library receives a reply from the servers, it does the opposite, i.e., it decrypts the encrypted fields using the corresponding schemes and keys.

*Server side* – The server-side handles different commands in different ways. The `out` command is executed the same way as in DepSpace. The fields may be encrypted but they come encrypted from the client so the tuple is stored unmodified. The `inp` and `rdp` commands were modified using DepSpace's extension mechanism in order to support the $=$, $<>$, $>$, $>=$, $<$, $<=$, and text search operations over encrypted data, returning one of the matching tuples. The `rdall` and `inall` commands work similarly, as `rdp` and `inp`, but return all matching tuples. The `rdSum` and `rdProd` commands are implemented as a modification of the original `rdAll` command that returns a single tuple with the relevant fields respectively added or multiplied.

## IV. Experimental Evaluation

We did a set of experiments to evaluate the performance of HomomorphicSpace. The experiments were executed in two personal computers. The first had an Intel(R) Core(TM) i7-3537U CPU @ 2.00 GHz, 4 GB RAM, and Windows 8.1 (64 bits). The second had an Intel(R) Core(TM)2 Duo CPU U9400 @ 1.40 GHz, 3,5 GB RAM, and Ubuntu 15.10 (64 bits). The software was executed using Java 1.8 with Oracle JDK in the Windows Machine and OpenJDK in the Linux Machine. The 2 machines were connected by an IEEE 802.11b/g/n switch (up to 54 Mbps).

We used the Linux machine to run the client, and the Windows machine to run the servers. Although we used a single machine for the server-side, it contained 4 server replicas. The client-side application had a *command line interface* that allows writing commands to be executed by the tuple space. The performance of tuple space operations depends on the

TABLE II
EXACT MATCH EXECUTION TIMES (MS)

| Encryption used | out 100 tuples | rdp 1 tuple | inAll |
|---|---|---|---|
| No encryption | 3659 ± 465 | 30 ± 5 | 235 ± 39 |
| Deterministic | 3747 ± 724 | 35 ± 5 | 342 ± 62 |
| Order Preserving | 3771 ± 580 | 32 ± 8 | 312 ± 75 |

load of the space, so we started all the experiments with an empty tuple space.

*1) Performance of tuple exact matching with encrypted fields:* In order to evaluate the performance of exact matching (equality) with encrypted fields for the relevant encryption schemes (and no encryption), we made the following test: (1) insert (`out`) 100 tuples with a single field in the tuple space, which are encrypted in the cases of Determinist and Order Preserving encryptions; (2) execute an exact match with `rdp value` and decrypt the tuple retrieved (if encrypted); (3) retrieve all tuples from the space with `inAll *` and decrypt the 100 tuples (if encrypted).

The tests made were all exact match (equality), independently of the encryption scheme or no encryption used (see Step 2 above). However, the performance for inequalities (different, greater than, greater or equal to, ...) would be very similar as all of them are simple byte comparisons. Each test was executed 30 times and the times for the three steps were measured. The results are in Table II. A first conclusion from the table is that encryption has no impact in the match operations, as the comparisons without encryption (2nd row) and with encryption (3rd and 4th rows) take very similar times (column for command `rdp`). A second conclusion is that the use of encryption (3rd/4th rows versus 2nd row) did not cause observable delay in the experiments (2nd and 4th columns). This result is consistent with the values obtained for the library, with encryption/decryption times that are fractions of a millisecond. Furthermore, the encryption/decryption load is *at the client*, not at the server side, so it has no impact in the capacity of the servers to process requests.

*2) Performance of the ordered operations:* In order to evaluate the performance of the ordered operations we made the following test: (1) insert (`out`) 100 tuples with a single field in the tuple space, with values from 0 to 99, encrypted with the Order Preserving scheme; (2) execute an `rdp` (read one matched tuple), with the parameters indicated in the first column of Table III and decrypt it; (3) execute an `inAll` (read and delete all matched tuples), with the parameters indicated in the first column of Table III and decrypt. The test was repeated 30 times.

We can observe in the table that the execution times of the `rdp` command are all inside the deviation intervals of each other, meaning that the type of match does not affect the execution times. For the `inAll` operation we can see that the slower operation in the one that reads all the tuples ($*$), the second slower is the one that reads all minus one (<>), and the faster operation is the one that reads just one tuple (=). The other operations have execution times somewhere in the middle. This allow us to conclude that the execution time of the `inAll` operation depends not on the type of comparison, but on the number of tuples retrieved. This is caused by the communication delay caused by more data.

TABLE III
ORDERED OPERATIONS EXECUTION TIMES

| Condition | rdp (ms) | inAll (ms) | Tuples selected |
|---|---|---|---|
| $*$ (match all) | 35 ± 8 | 257 ± 27 | 100 |
| = (match) | 29 ± 9 | 33 ± 22 | 1 |
| <> 50 | 28 ± 6 | 209 ± 7 | 99 |
| < 50 | 31 ± 7 | 195 ± 46 | 50 |
| <= 50 | 30 ± 8 | 174 ± 21 | 51 |
| > 50 | 29 ± 7 | 181 ± 26 | 49 |
| >= 50 | 34 ± 8 | 204 ± 53 | 50 |

*Acknowledgements* This work was supported by the European Commission through project H2020-653884 (SafeCloud) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

## REFERENCES

[1] Cloud Security Alliance, "The notorious nine: Cloud computing top threats in 2013," Feb. 2013.
[2] F. Rocha and M. Correia, "Lucy in the sky without diamonds: Stealing confidential data in the cloud," in *Proc. 1st DCDW Workshop*, 2011.
[3] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annual ACM Symposium on Theory of Computing*, 2009.
[4] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology – EUROCRYPT 2010*. Springer, 2010.
[5] M. Yagisawa, "Fully homomorphic encryption without bootstrapping," Cryptology ePrint Archive, Report 2015/474, 2015.
[6] K. Lauter, M. Naehrig, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proc. 3rd ACM Workshop on Cloud Computing Security*, 2011.
[7] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Proc. 28th Annual International Conference on Advances in Cryptology*, 2009.
[8] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, Jan 2014.
[9] B. Ferreira, J. Rodrigues, J. Leitão, and H. Domingos, "Privacy-preserving content-based image retrieval in the cloud," *CoRR*, vol. abs/1411.4862, 2014.
[10] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *Computer Security - ESORICS*, 2015.
[11] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. 23rd ACM Symposium on Operating Systems Principles*, 2011.
[12] M. Burrows, "The Chubby lock service for loosely-coupled distributed systems," *Proc. 7th Symposium on Operating Systems Design and Implementation*, 2006.
[13] J. Baker, C. Bond, J. Corbett, and J. Furman, "Megastore: Providing scalable, highly available storage for interactive services." in *Proc. 5th Biennal Conference on Innovative Data Systems Research*, 2011.
[14] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free coordination for Internet-scale systems," in *Proc. 2010 USENIX Annual Technical Conference*, 2010.
[15] GigaSpaces, "XAP 9.0 documentation – product overview – concepts," http://wiki.gigaspaces.com/wiki/display/XAP9/Concepts, 2011.
[16] A. N. Bessani, E. P. Alchieri, M. Correia, and J. S. Fraga, "DepSpace: a Byzantine fault-tolerant coordination service," in *Proc. 3rd ACM SIGOPS/EuroSys European Systems Conference*, Apr. 2008.
[17] T. Distler, C. Bahn, A. Bessani, F. Fischer, and F. Junqueira, "Extensible distributed coordination," in *Proc. 10th ACM SIGOPS/EuroSys European Systems Conference*, 2015.
[18] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programing Languages and Systems*, vol. 7, no. 1, Jan. 1985.