Journal of Internet Services and Applications

RESEARCH Open Access

Providing privacy on the tuple space model

Edson Floriano[1], Eduardo Alchieri[1]*, Diego F. Aranha[2] and Priscila Solis[1]

Abstract

Conceptually, tuple spaces are shared memory objects that provide operations to store and retrieve ordered sets of data, called tuples. Tuples stored in a tuple space are accessed by the contents of their fields, working as an associative memory. Although there are some proposals for secure tuple spaces, accessing tuples through field contents makes these systems susceptible to attacks that could impair user and data privacy, since servers must access tuple data. In order to deal with these limitations and provide privacy in the tuple space model, this paper proposes some extensions to DepSpace, a tuple space system that implements dependability and security properties through a set of mechanisms that are not enough to ensure privacy. The resulting system provides privacy and, at the same time, allows tuple selection/matches similar to the traditional insecure model, i.e., it does not constraint the matching possibilities. The main problem to be addressed is that servers must select tuples based on their contents without knowing them. The proposed solution uses robust cryptographic schemes, as order-preserving encryption and homomorphic encryption, to provide this functionality without revealing the tuple contents. An analysis concerning security aspects of DepSpace is presented, as well as the benefits of the proposed solutions. A set of experiments, executed with an implementation of the proposed protocols, shows the feasibility of the proposed solutions and shed some light on both the behavior of the system and the costs to provide privacy in the tuple spaces model.

Keywords: Tuple space, Privacy, Searchable encryption, Homomorphic encryption

1 Introduction

The distributed computing community has given a lot of attention to the security issues on the design and development of distributed applications. A system is secure if it satisfies the confidentiality, availability and integrity properties [1]. Furthermore, one can intuitively understand privacy, under the perspective of some entity, as the confidentiality of its sensitive information (data and metadata) [2]. This entity may be a person, an organization, a nation, etc. Therefore, privacy is closely connected to the confidentiality of information.

Currently, there are many factors that increase the risk related to the security of applications [2]: *(i)* the world is becoming a huge infrastructure, interconnected and interdependent; *(ii)* there are massive amounts of correlated data available; *(iii)* the entities are exposing themselves much more; and *(iv)* the number of software vulnerabilities is increasing.

In face of this scenario, many systems aim to provide information confidentiality by protecting only the secret data itself, without any care of the correlated non-confidential data. However, statistical inference attacks [3] often are able to recover secret information from the analyses and correlation of public available data. Consequently, it becomes interesting, if not mandatory, to develop solutions and provide means to protect all information handled by secure applications.

These aspects are particularly relevant when we consider data shared through a tuple space [4, 5], since in this model the accesses are done by the tuple contents, working as an associative memory. As examples of distributed systems that could benefit from a secure tuple space, it is possible to mention either high-level applications such as secure biddings [6], and applications that need a distributed shared memory [6, 7] or synchronization building blocks as shared counters and distributed lists [8].

*Correspondence: alchieri@unb.br
[1]Department of Computer Science, University of Brasilia, UnB, Brasília, DF, Brazil
Full list of author information is available at the end of the article

Although there are some proposals to secure tuple spaces [5, 8–11], the fact that the tuples are accessed by their contents makes these systems susceptible to attacks that could impair the privacy of the data and, consequently, of their users. The main problem to be addressed in order to provide privacy in this model is that servers must select tuples based on their contents without knowing them.

Among the existent proposals, DEPSPACE [5] is the system that provides the higher security level, employing both access control and cryptographic mechanisms. This system suggests the classification of the tuple fields as follows: public – the data is public and all parties/processes can access it; comparable – a hash of the field content is available, consequently, if the range of values that a field can take is known and limited, then a preimage attack can disclose its contents; and private – no information is available. Given this classification, at least one field of each tuple needs to be public or comparable to allow tuple accesses, i.e., to allow servers to execute matches among tuples and templates (Section 2). This approach, although adequate since it provides some level of confidentiality, brings a big challenge to the development of applications: if a lot of public and/or comparable fields are used, then the system becomes vulnerable to correlation, preimage and collision attacks; otherwise, as servers are not able to execute searches/matches in private fields, the tuple searches/matches possibilities are significantly reduced, limiting its use in the development of distributed applications.

In order to circumvent these problems, this paper proposes extensions to the DEPSPACE field classification to prevent that the privacy of data and users are breached by attackers. Through the use of robust and modern cryptographic mechanisms, the proposed extensions preserve the security properties and, at the same time, allows more flexibility in the execution of tuples searches/matches. Moreover, this paper presents an analysis about the security provided by DEPSPACE, as well as about the resulting system after the inclusion of the proposed solutions.

In summary, this paper makes the following contributions:

- It increases the security provided by DEPSPACE, mainly privacy, by proposing new field classification that uses robust cryptographic schemes and, consequently, reduces (and even eliminates) the need for public or comparable fields. Additionally, the proposed new fields bring more flexibility in the execution of tuples searches/matches since they use searchable encryption schemes.
- It presents an analysis of the security provided by DEPSPACE prior and after the inclusion of the proposed solutions.
- It analyzes, through a set of experiments, the impact on the system performance caused by the suggested extensions.
- It discusses some relevant aspects around such extensions, like other implementation possibilities and current limitations.

The remainder of this paper is organized as follows. Section 2 details the concept of tuple space and introduces DEPSPACE, analyzing the security properties provided by this system. Section 3 discusses robust cryptographic schemes that are used in the proposed extensions, which are presented at Section 4. Section 5 discusses the integration of the proposed solutions within DEPSPACE. An experimental evaluation about the proposed solutions is presented at Section 6. Section 7 brings some important discussions about the proposed solutions. The related works are discussed at Section 8. Finally, conclusions and future work are given in Section 9.

2 Tuple space

Conceptually, a tuple space can be seen as a shared memory object that provides operations to store and to retrieve ordered data sets, called tuples. Processes in a distributed system can then interact through this shared memory abstraction. A tuple is an ordered sequence of fields, where a field that contains a value is said to be defined. A tuple t where all the fields are defined is called entry (or tuple). A tuple $\bar{t}$ is called template if any of its fields does not have a defined value. A tuple t and a template $\bar{t}$ combine (or match) if, and only if, both has the same numbers of fields and all the values and types of the defined fields in $\bar{t}$ are identical to the values and types of the corresponding fields in t. For example, a tuple $\langle \text{JISA}, 2017, \text{SBC} \rangle$ combines/matches with the template $\langle \text{JISA}, *, * \rangle$ ('$*$' denotes a undefined field, called wildcard).

Process coordination through tuple spaces, introduced by the programming language LINDA for parallel systems [4], supports decoupled communications in space (processes do not need to know each other locations) and in time (processes do not need to be active at the same time). Besides that, this model of coordination provides some synchronization power.

Manipulations performed in tuple spaces consist in invocations of three basic operations [4]: $out(t)$ that stores the tuple t in the space; $in(\bar{t})$, that removes from the space a tuple that matches the template $\bar{t}$; $rd(\bar{t})$, used to read from the space a tuple that matches the template $\bar{t}$, without removing it. Operations *in* and *rd* are blocking, i.e., if there is no tuple that matches the template in the space, the process gets blocked until one is available. A common extension to this model is the inclusion of non-blocking variants of these operations, denominated *inpn* and *rdp*. These operations work exactly like the previously, except by the fact that they return even if there is not a tuple that matches the template (indicating its nonexistence).

Another operation implemented in some tuple spaces (e.g., DEPSPACE [5]) is the $cas(\bar{t})t$ (conditional atomic swap) [12, 13]. This operation works like an atomic execution of the code: **if** $\neg$ $rdp(\bar{t})$ **then** $out(t)$ ($\bar{t}$ is a template and t an entry/tuple). The operation inserts t in the space iff $rdp(\bar{t})$ does not return any tuple, i.e., if there is no tuple in the space that matches $\bar{t}$; otherwise it returns a tuple that matches $\bar{t}$.

Notice that according to the previous definitions, tuple spaces work as an associative memory: tuples/data are accessed by their contents, not by their addresses. Figure 1 illustrates the $out(t)$, $rdp(\bar{t})$ and $inp(\bar{t})$ operations, showing the operation sent by the client, the servers replies and the final state of the tuple space.

2.1 DEPSPACE: A BFT coordination system

The DEPSPACE [5] system provides a Byzantine Fault-Tolerant (BFT) [14] coordination service based on the tuple space model. The following security and dependability attributes (or properties) are necessary for this model [1]: *(1)* reliability – the operations executed in the tuple space change its state according to their specification; *(2)* availability – the tuple space is always ready to execute the operations required by authorized parties; *(3)* integrity – no improper alteration of the tuple space can occur; *(4)* confidentiality – the content of tuple fields cannot be disclosed to unauthorized parties. With the goal of ensure these properties, DEPSPACE is built over a set of layers, each one responsible for the execution of a different functionality.

2.1.1 DEPSPACE *layers*

This section introduces the DEPSPACE layers emphasizing the confidentiality layer, which is responsible for the aspects related to this work. Figure 2 shows the layers and their location in the stack at both clients and servers.

Replication. To maintain consistency in the tuple space, DEPSPACE utilizes State Machine Replication [15, 16] as the bottom layer. This mechanism is related mainly with the properties of availability, integrity and confidentiality. Considering a system with n replicas/servers, it ensures that operations are executed according to their specification even if up to $f = (n-1)/3$ replicas are malicious (the correct replicas mask the behavior of the malicious ones). Through these protocols, the correct replicas execute the same sequence of operations and returns the same values, evolving in a synchronized way.

Confidentiality. Since tuples are maintained replicated in a set of servers, the provision of confidentiality (and privacy) must not be attributed to a single server because up to f of them could fail and expose the tuple contents to unauthorized parties.

Consequently, DEPSPACE implements confidentiality through the use of a $(n,f+1)$-Publicly Verifiable Secret Sharing (PVSS) [15] scheme. The clients, which represent the dealers in the scheme, generate a secret that they use to encrypt the tuples. Later, they generate a set of n shares of this secret and one different share is sent to each server. The secret can be recovered only with a combination of $f+1$ shares, what makes it impossible for a collusion of up to f malicious servers to expose the tuple contents.

As servers cannot access the tuple contents (since they are encrypted by the client), the protocol employs a *fingerprint* for the tuple, making it possible to implement and compute the matches between tuples and templates at the servers. The fingerprint is computed according to the type of each tuple fields, which can be classified as follows:

- **Public (PU)**: the field content itself is used as its fingerprint, i.e, no cryptographic method is applied to the field content and it remains exposed.
- **Comparable (CO)**: a hash of the field content is used as its fingerprint (using a collision resistant hash function), allowing servers to execute

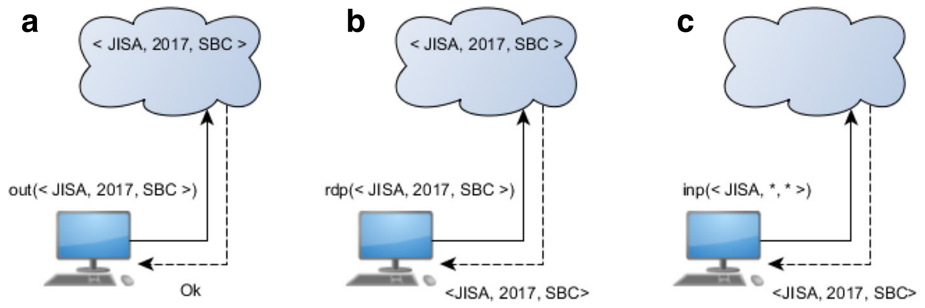

Fig. 1 Tuple space basic operations. **a** *out*(t): servers receive a tuple t from the client, store t in the tuple space and return "ok". **b** *rdp*($\bar{t}$): servers receive a template $\bar{t}$, find a tuple t that matches $\bar{t}$ and return t keeping it on the tuple space. If no tuple is found, servers return null. **c** *inp*($\bar{t}$): servers receive a template $\bar{t}$, find a tuple *t* that matches $\bar{t}$ and return t removing it from the tuple space. If no tuple is found, servers return null

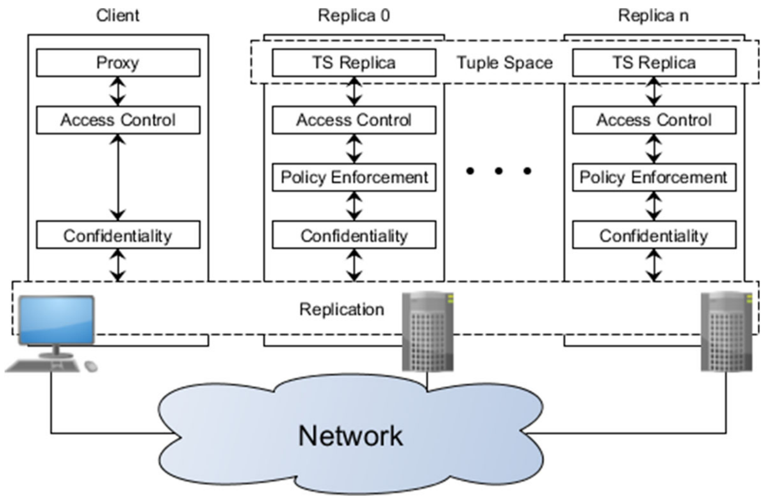

Fig. 2 DepSpace layers

searches/matches in these type of fields while, at the same time, providing some level of security.

- **Private (PR)**: a special symbol (PR) is used as *fingerprint* of these fields. Although it provides a level of security higher than the CO classification, no information in this field is available at the servers to verify if a tuple matches a template.

Once it is not possible to send different versions of a request for different servers in the state machine replication approach (containing only its share of the secret used to encrypt the tuple), the client encrypts each share with a secret key shared with the server that will store it. Consequently, each server will access only its share and, as a malicious server does not have access to all shares, it can not restore and expose the tuples contents.

In a nutshell, a insertion operation (*out*) works as follows:

- The client generates a secret s and encrypts the tuple using this secret.
- The client uses the PVSS scheme to generate n shares of s.
- The client encrypts each share with a secret key shared with each server (one share per server).
- The client computes the fingerprint according to the fields classification.
- The client uses the state machine replication protocol to send a request to the servers (in this protocol it must wait for $f + 1$ replies to finish the request execution). The request contains the encrypted tuple, the encrypted shares, the proof that these shares are valid and the tuple fingerprint.
- When a server executes this request, it only stores all received data and sends an acknowledge to the client as a reply.

On the other hand, the protocol to read/remove a tuple works as follows:

- The client computes the fingerprint for the template according to the field classification. The fingerprint of a undefined field is the wildcard itself.
- The client uses the state machine replication protocol to send a read/remove operation to the servers containing the generated fingerprint.
- When a server executes this request, it chooses a tuple deterministically such that its fingerprint matches the received fingerprint (if it is a removal operation, this tuple is removed from the space). In case its share was not yet verified, it extracts its share and verify if this share is valid using the proofs received during the *out* operation. Afterward, the server replies to the client with the encrypted tuple, its encrypted share (to avoid eavesdropping on the replies), the tuple fingerprint and proofs that the share is valid.
- The client waits for $f + 1$ replies, decrypts the shares, verifies their validity and combines them to recover the secret s. Finally, the client decrypts the tuple using s.
- The client verifies if the fingerprint it used is valid for the recovered tuple. If the fingerprint is valid, the operation is finished. Otherwise, a repair procedure is executed to remove invalid data from the space and the operation is repeated.

Notice that, according to the fingerprint definitions, searches are possible only in public and comparable fields, i.e., private fields cannot be used to verify if a tuple matches a template and are always used as undefined fields on the template. This limitation brings at least two consequences. On the one hand, a tuple with many private fields makes the search very restricted, losing the flexibility in the development of applications because a template with many undefined fields does not allow a fine-grained match at the servers. On the other hand, a tuple with many public and/or comparable fields is susceptible to many attacks, like correlation and preimage attacks.

Policy enforcement. This layer allows a fine-grained access policy execution [7] that takes into account three parameters (identifier of the invoker, operation and arguments, and the current tuples stored in the space) to decide if an operation is approved or denied. These policies are defined by the users and are loaded at the servers during the system setup.

Access control. Access control is a fundamental mechanism to keep the integrity and confidentiality of information (tuples) stored in the DepSpace since it prevents unauthorized clients from getting access to the tuples. Moreover, this mechanism prevents malicious clients from saturating the tuple space by sending a lot of tuples. Currently, the DepSpace implements the access control based on credentials: for each tuple inserted in the DepSpace, a set of credentials are necessary to access it, both to read and to remove it from the space (access control at tuple level). These credentials are defined by the process that inserts the tuple. Moreover, it is possible to define which credentials are necessary to insert a tuple into the space (access control at space level) during its setup. The implementation of this functionality is realized through the association of access control lists to each tuple and space.

2.1.2 Security analysis

Below we briefly summarize some security definitions. According to [17], the attacks against the cryptographic schemes aim to obtain the plaintext or the decryption key through the following methods:

- *Ciphertext-only attack (COA)*: In this kind of attack, an adversary tries to obtain the decryption key or the plaintext only having the ciphertext at its disposal. This is the weaker type of attack and, therefore, a system vulnerable to this attack is considered insecure.
- *Known-plaintext attack (KPA)*: In this attack, the adversary has at its disposal a significant amount of plaintexts and their corresponding ciphertexts. Through the comparison of plaintexts and their corresponding ciphertexts, the adversary tries to discover the decryption key or to decrypt another ciphertext.
- *Chosen-plaintext attack (CPA)*: the adversary chooses a plaintext and receives the corresponding ciphertext for analysis, which may allow him/her o discover the plaintext corresponding to another ciphertext.
- *Adaptive chosen-plaintext attack (CPA2)*: This attack is similar to CPA, however the attacker can choose new plaintexts depending on the received answer.
- *Chosen-ciphertext attack (CCA)* In this kind of attack, the adversary chooses a ciphertext and receives (without access to the decryption key) the corresponding plaintext. The adversary uses the analysis of this correlation to discover the plaintext corresponding to another ciphertext.
- *Adaptive chosen-ciphertext attack (CCA2)*: This attack is similar to the CCA, however the attacker can choose new ciphertexts depending on the received answer. This attack is considered very strong and much harder to implement.

The attacks above are presented in order of increasing complexity. A system vulnerable to a weak attack will be classified at a lower security level, even if it resists a stronger attack. Although these are the main attacks considered in the literature, many other attacks could be possible depending on the system characteristics. For example, in [3] the authors show that it is possible to perform *inference attacks* by means of correlation of the ciphertexts with additional public information. In this case, if there is a strong correlation between the encrypted and the public data, the plaintexts could be recovered with high accuracy. Considering encrypted databases of hospitals, [3] presented a study in which more than 60% of the data deterministically (Section 3.1) encrypted (e.g.: sex, race and mortality risk) could be discovered in 60% of the hospitals, while more than 80% of data encrypted with order preserving (Section 3.2) encryption (e.g.: age and disease severity level) were recovered in 95% of the hospitals.

As in practice it is impossible to achieve total security against these attacks for all the *mathematically possible* adversaries, a weaker security definition is necessary, taking into account only the *computationally possible* adversaries. In this context, a system is defined informally as *semantically secure* if it is able to resist, with high probability, to attacks performed by any adversary computationally efficient [18]. Based on the formal definitions of [19], we define informally that for any efficient adversary $\mathcal{A}$, a cipher $E = (E, D)$ defined over (K, M, C) offers:

- *Indistinguishability against chosen-plaintext attacks (IND-CPA)*: the cipher offers IND-CPA if for all attempts $i = 1, 2, ...q$, given two messages

$m_{i0}, m_{i1} \in \mathcal{M}$ of the same size, chosen by $\mathcal{A}$ and submitted to an oracle that answers with the ciphertext $c_i = E(k, m_{ib}) \in \mathcal{C}$ for some key k selected randomly in $\mathcal{K}$ and $b \in \{0, 1\}$, the probability that $\mathcal{A}$ can distinguish between $c_i = E(k, m_{i0})$ or $c_i = E(k, m_{i1})$ is negligible.

- *Indistinguishability against chosen-ciphertext attacks (IND-CCA)*: the cipher offers IND-CCA if, for the same conditions of the IND-CPA, the adversary $\mathcal{A}$ also can get access to a oracle that given a ciphertext $c_i \notin \{c_1, ..., c_{i-1}\}$ answers with the corresponding plaintext $m_i = D(k, c_i)$ and, in the same way, the probability that $\mathcal{A}$ can distinguish between $c_i = E(k, m_{i0})$ or $c_i = E(k, m_{i1})$ is negligible. In this case, $\mathcal{A}$ can make as many requests as it wants to the decryption oracle, however only until it has received the challenge ciphertext from the encryption oracle.
- *Indistinguishability against adaptive chosen-ciphertext attacks (IND-CCA2)*: the cipher offers IND-CCA2 if, besides the conditions established to the IND-CCA, the adversary can continue using the decryption oracle even after it had received the challenge cryptogram. The only restriction is that it is not allowed to submit this cryptogram for decryption.

Additionally, we have the following IND-CPA relaxations for both deterministic (Section 3.1) and order-preserving (Section 3.2) ciphers, respectively:

- *Indistinguishability against distinct chosen-plaintext attacks (IND-DCPA)*: the cipher $\mathcal{E}$ offers IND-DCPA if it is deterministic and for all attempts $i = 1, 2, ..., q$, given two messages $m_{i0}, m_{i1} \in \mathcal{M}$ of the same size chosen by $\mathcal{A}$, distinct for each attempt ($\forall i, j \in \{1, 2, ..., q\}, m_{i0} \neq m_{j0}$ and $m_{i1} \neq m_{j1}$), submitted to the oracle that answers with the ciphertext $c_i = E(k, m_{ib}) \in \mathcal{C}$ for some key k selected randomly in $\mathcal{K}$ and $b \in \{0, 1\}$, the probability that $\mathcal{A}$ can distinguish between $c_i = E(k, m_{i0})$ or $c_i = E(k, m_{i1})$ is negligible [19].
- *Indistinguishability against ordered chosen-plaintext attacks(IND-OCPA)*: the cipher $\mathcal{E}$ offers IND-OCPA if it preserves the order between the plaintexts and for all attempts $i = 1, 2, ..., q$, given two messages $m_{i0}, m_{i1} \in \mathcal{M}$ of the same size, chosen by $\mathcal{A}$ and submitted always in the same order (i.e., $m_{i0} < m_{j0} \iff m_{i1} < m_{j1}$ for all $1 \leq i, j \leq q$) to an oracle that answers with the ciphertext $c_i = E(k, m_{ib}) \in \mathcal{C}$ for any key k selected randomly in $\mathcal{K}$ and $b \in \{0, 1\}$, the probability that $\mathcal{A}$ can distinguish between $c_i = E(k, m_{i0})$ or $c_i = E(k, m_{i1})$ is negligible [20].

Using these definitions, it is possible to highlight some vulnerabilities of DEPSPACE. The main focus for the investigation is the way the fingerprint is generated. In the following we discuss the vulnerabilities related with comparable and public fields classifications:

- Comparable fields allow tuple selection/matches without servers knowing the field contents, but the use of hash functions makes the system vulnerable to collision and preimage attacks. In fact, an adversary is able to get a desired amount of inputs and their respective outputs by calculating their hashes. Consequently, if the set of values that a comparable field could assume is small and known, then the attacker can calculate the hashes for all possible values, learning the correspondence between plaintexts and ciphertexts. This attack is similar to the known-plaintext attack, except for the fact that in this case there is no encryption and decryption functions.
- Public fields are not subject to a disclosure attack since their contents are already public. However, these fields could provide useful information to an attacker, which could correlate the encrypted tuple contents with a public database and execute an inference attack. Comparable fields also could be used for these attacks since their contents could be inferred.

3 Robust cryptographic schemes

In order to circumvent the aforementioned limitations and vulnerabilities, this section introduces some robust cryptographic schemes that allow searches and computations over encrypted data. These schemes were used to improve DEPSPACE security. Based on its characteristics, mainly the way fingerprints work, we looked for cryptography schemes that best fit this system.

3.1 Deterministic and probabilistic ciphers

A cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ is called *probabilistic* if, for a fixed key $k \in \mathcal{K}$ and messages $m \in \mathcal{M}$ that are used as inputs of the encrypt function $E : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$, the output $c = E(k, m)$ may assume different values. Otherwise, the cipher is said to be *deterministic* [18].

Naturally, an inherent characteristic of deterministic ciphers is the leakage of equality of texts encrypted under the same key, that is, $m_0 = m_1 \iff E(k, m_0) = E(k, m_1)$. This fact could be used to perform equality searches over encrypted data [21, 22]. Clearly, this kind of cipher does not achieve security against the Chosen-Plaintext Attack (CPA) since an adversary may submit in the beginning two copies of the same message m_0 to the oracle and receive two identical ciphertexts $c_0 = E(k, m_0)$. Afterwards, the attacker could send the messages m_0 and m_1 to the oracle, receiving $c_b = E(k, m_b)$, where $b \in \{0, 1\}$. Now, it is enough to compare c_b ($b \in \{0, 1\}$) with c_0 to know if it is $E(k, m_0)$ or $E(k, m_1)$.

Probabilistic ciphers can resist stronger attacks, achieving the IND-CCA2 security level when combined with authentication primitives.

3.2 Order-preserving encryption - OPE

A symmetric *order-preserving* cryptography scheme preserves the order relation among the encrypted values [20]: for all i and j and for all keys k, $E(k,i) < E(k,j) \iff i < j$. This kind of cipher leaks the order among plaintexts through the ciphertexts, therefore not achieving the IND-CPA security level. Observing that this security level is not achievable by a deterministic algorithm with such property, even considering a weak security level (IND-OCPA), applying pseudo-random generators and functions (PRF's and PRG's) was proposed [20]. This approach provides a flexible (but strong) level of security called *pseudorandom order-preserving function under chosen-ciphertext attack* (POPF-CCA).

However, this security definition does not determine what kind of data could leak, besides the order. In [23], *Order Revealing Encryption (ORE)* is proposed, a construction that minimizes the amount of leaked data. Although this approach presents a higher level of security, it is impractical since the initial proposal presents poor performance. Trying to circumvent this limitation, a practical ORE algorithm [24] was proposed. This algorithm achieves the IND-OCPA security level since it could leak only the first bit that differs in the compared values. Finally, an ORE algorithm that resists inference attacks [25] was proposed and presents a good performance for encrypted database applications [22].

3.3 Homomorphic encryption

Homomorphic encryption [26] allows computations over encrypted data without decrypting them and without the knowledge of the secret keys. Given any two messages, m_1 and m_2, a homomorphic encryption function E and a key k, we have that $E(k,m_1) \circ_c E(k,m_2) = E(k,m_1 \circ_m m_2)$, where $\circ_m$ denotes an arithmetic operation on the message domain and $\circ_c$ denotes an arithmetic operation on the ciphertext domain.

Fully homomorphic systems generally present poor performance and are not practical in the development of applications. However, it is possible to avoid the performance issues of fully homomorphic schemes while preserving some of their functionality, since many applications need only some kind of operations, what can be done by a "somewhat" homomorphic scheme [27]. These schemes present a better performance and are practical. Paillier [28] and exponential ElGamal [29] are examples of efficient "somewhat" homomorphic ciphers. These schemes are also probabilistic, achieving the IND-CPA security level.

4 Providing privacy on DepSpace

This section presents our proposal to increase security of the DepSpace system. In a nutshell, new types for field classifications are introduced, which use the previously discussed robust cryptographic schemes to provide the same functionality as the originally proposed DepSpace (or even bringing more flexibility to the development of applications since these fields allow the implementation of previously impossible secure searches), however, with stronger security properties.

Before presenting the new field classifications, let us introduce another very important characteristic of DepSpace. This system uses a $(n, f+1)$-PVSS scheme [15] to split a secret key, that is used to encrypt some tuple (Section 2.1), among n servers, requiring $f+1$ of them to recover the secret key. This approach works fine since at most f malicious servers are supposed in the system. However, to avoid such compromise, these keys must be known only to the clients, that must previously share it utilizing a public-key cryptographic algorithm with a mechanism that provides public key verification and protection against "man-in-the-middle" attacks [30]. Independent groups of processes that must communicate through the space can have a different shared key. This protocol is orthogonal to the way fingerprints work in DepSpace, which is the focus of this paper.

4.1 Improving fingerprint security

The analysis presented in Section 2.1.2 showed that DepSpace is subject to many simple attacks. To circumvent this problem, we propose the reduction (and even elimination) in the use of public and/or comparable fields and the adoption of the following ones to create the fingerprints:

- **Comparable deterministic (CD)**: the field content f is encrypted through a deterministic symmetric encryption algorithm by using a function *encryptCD*(key_{shared}, f). The resulting ciphertext is used as fingerprint and, as the algorithm is deterministic, it allows servers to execute searches/matches in these fields. Notice the same key is used to decrypt these fields.
- **Operable (OP)**: the field content f is encrypted through a homomorphic or "somewhat" homomorphic asymmetric encryption algorithm by using a function *encryptOP*(key_{public}, f). The resulting ciphertext is used as fingerprint and allows computations at the servers. Notice that this scheme is asymmetric: the public key is used for encryption and computations over f; the private key is used to decrypt the result.
- **Orderly (OR)**: the field content f is encrypted through an order-preserving symmetric encryption

algorithm, as OPE, by using a function *encryptOR*(key_{shared},f). The resulting ciphertext is used as fingerprint, allowing the execution of matches and the ordering among these fields. Notice the same key is used to decrypt these fields.

The fingerprint function ensures that if a tuple t matches a template $\bar{t}$, the fingerprint t_h of t matches the fingerprint $\bar{t}_h$ of $\bar{t}$ if both are generated using the same protection type (field classification) for each field [5]. The fingerprint $t_h = \langle h_1, ..., h_m \rangle$ of a tuple $t = \langle f_1, ..., f_m \rangle$ is generated according to the formula presented at Fig. 3.

The new field classification brings a key management issue since it is necessary to share a secret key for CD and OR fields or a key pair (public and private) for OP fields. These keys must be shared among the processes that are communicating through the tuple space. For OP fields, the public key also is available to the servers allowing they to execute computations over these fields. Fortunately, key management is not a problem in our model since the tuple space itself could be used for this coordination (Section 7.3).

CD fields. We strongly recommend the preference of CD fields over CO. By using a deterministic symmetric cipher instead of a hash value, an attacker would need to access the secret key to encrypt a field, making it impossible to mount a preimage attack (Section 2.1.2). However, the CD classification must be employed carefully since this cipher reveals if plaintexts are equal. Consequently, if the field content belongs to a small domain, using few external information is enough to disclose it. For instance, consider the encryption of a field that contains the sex in a database that is known to have more men than women. An attacker could observe that there are only two possible ciphertexts and conclude that the one with more occurrences refers to the male sex. Therefore, this cipher should be used for fields that store indexes, with a high amount of possible values, and for non-sensitive data like identifiers, e-mail address, name of process nodes, among others.

Some cryptographic algorithms use operation modes with randomized initialization vectors (IV), in the form $c = E(k, m, IV)$, as a way to provide probabilistic encryption. To provide deterministic encryption, these algorithms usually fix the IV (e.g. in zero). In these cases, we recommend the use of a PRF (pseudo random function) over the message m by using a key k_1, producing a pseudo-random output $r = F(k_1, m)$. The encryption function then uses r as IV and another key k_2 to produce the encrypted output $c = E(k_2, m; r)$. Since the PRF generates the same output for the same message and key, the algorithm remains deterministic. Moreover, the PRF generates different outputs for different messages inputs (even under the same key) and, therefore, different IV's are obtained for different messages, achieving the IND-DCPA security level [18].

$$h_i = \begin{cases} * & \text{if } f_i = * \\ f_i & \text{if } f_i \text{ is PU} \\ \mathsf{hash}(f_i) & \text{if } f_i \text{ is CO} \\ \text{PR} & \text{if } f_i \text{ is PR} \\ \text{encryptCD}(key_{shared},\ f_i) & \text{if } f_i \text{ is CD} \\ \text{encryptOP}(key_{public},\ f_i) & \text{if } f_i \text{ is OP} \\ \text{encryptOR}(key_{shared},\ f_i) & \text{if } f_i \text{ is OR} \end{cases}$$

Fig. 3 Fingerprint generation

OP fields. OP fields allow the computation over encrypted data. For these fields, a homomorphic or a "somewhat" homomorphic cipher (Section 3.3) should be used, such as Paillier [28], according to the application requirements. These fields increase the functionality provided in DEPSPACE. For instance, it is possible to update values used to synchronize decoupled process without revealing process status.

If the field needs only one kind of arithmetic operations (e.g.: addition/subtraction or multiplication/division), a "somewhat" homomorphic cipher can be used to offer better performance than a fully homomorphic alternative.

Notice that all fingerprint fields are used only to select tuples, except for OP fields. In these cases, the updated value to be read is in the fingerprint, not in the decrypted tuple, and the client must consider this value during a reading/removal operation.

OR fields. This classification brings a lot of functionalities for the tuple spaces allowing servers to execute some operations, as *(1)* tuples ordering based on a field, *(2)* execution of queries for a field belonging to some range and *(3)* select a tuple with a field storing the maximum/minimum value. To execute these operations in the original DEPSPACE system, fields should be classified as PU, losing security. If they are classified as PR, clients need to read and decrypt all tuples prior to perform these operations. Allowing servers to execute such processing improves the system performance since fewer data are transferred through the network [8].

The OR fields must be chosen carefully since these fields are vulnerable to inference attacks if all possible values of a domain are present in the data collection [3]. For instance, if a tuple field refers to the age of patients in a hospital that is known to have patients of all ages from zero to 100 years old, then data could be revealed due to this association.

To overcome this vulnerability, we suggest the use of a composition of algorithms. First, encrypt the data with an OPE algorithm [20] and later use the output as input of an ORE algorithm [24]. By this approach the system achieves the IND-OCPA security level, with the leakage of the first bit that differs in the compared values. The distance between OPE encrypted numbers is a random value and it does not have any connection with the distance of the original numbers. This is the reason for the first step since in this case the leaked bit is from the OPE encrypted number instead of the original number.

Security analysis. Table 1 shows the intrinsic security level for the ciphers used in each field type, ordered by the lower to the higher security level. It is important to remember that both IND-OCPA and IND-DCPA represent weaker notions of security based on IND-CPA. Although the OR fields reveal more than equality (the order among the ciphertext is also revealed), the IND-OCPA level is considered more secure than IND-DCPA by the fact that it is not achieved by deterministic algorithms.

Finally, the security of a system depends on the correct use of each cipher, considering the characteristics of the data stored in each field. Keeping all data encrypted, it is possible to avoid inference attacks that could cause a big damage to applications using lower computational effort.

5 Implementation

We used the original DEPSPACE implementation [5] and introduced some modifications to apply the previously discussed robust cryptography primitives. Basically, the tuple space operations still work as in the original system (e.g., the PVSS scheme was unchanged), but we extended the system by providing new possibilities in the fingerprint generation. In the following we discuss how the fingerprint is processed according to each field classification.

- Public (PU): no cryptography is used in these fields since their original contents are used in the fingerprint (plaintext).
- Comparable (CO): to process these fields we used the SHA-1 algorithm, which generates a hash output of 20 bytes.
- Private (PR): no original field information is used in the fingerprint, only a special symbol PR indicating its classification.
- Comparable Deterministic (CD): for these fields we used a HMAC-SHA256 (Hash-based Message Authentication Code with SHA-256) algorithm and a secret key of 256 bits to generate an encrypted output of 32 bytes.

Table 1 Security level for each field type

PU	CO	CD	OR	OP	PR
Insecure	Preimage/collision	IND-DCPA	IND-OCPA[a]	IND-CPA	IND-CCA2

[a]Level achieved if an ORE method is applied

- Operable (OP): for these fields we used the *javallier* library [31], a Java implementation of the Paillier algorithm [28]. This "somewhat" homomorphic encryption library implements the addition operation between two encrypted numbers, from which the subtraction can also be derived. Additionally, an encrypted number could be multiplied by a small plaintext number using repeated addition operations. For the asymmetric algorithm we used a key pair (public and private) of 512 bits.
- Orderly (OR): for these fields we used the *jope* library [32], a Java implementation of an OPE algorithm [20]. We employed this library without the composition with a ORE algorithm (see Section 3.2). This library outputs a deterministic BigInteger that preserves exactly the same order than the non encrypted number, with pseudo-random distances between any two encrypted numbers. To perform inequality queries (e.g.: less than or greater than), we implemented at the proxy layer of the client side a detector to identify the presence of these queries and handle them in a way that the tuple space layer at the servers side can understand, i.e, the tuple space layer was modified to perform matches using these queries. For instance, a tuple field containing the number 10 matches a template field containing a query "less than 11". The current supported queries are: lt (less than), le (less than or equal to), eq (equal to), gt (greater than), and ge (greater than or equal to).
  Considering the previously example, the OR tuple field contains the encrypted number 10 while the OR template field contains a query $lt(11)$ (less than 11).

Since these solutions allow computations at the servers, it is possible to implement secure *extensible distributed coordination services* [8], like shared counters, distributed queues and distributed barriers. Figure 4 shows a representation of DEPSPACE layers with an additional layer, called *Extension Manager*, to deal with operations in OP and OR fields according to applications need. The figure shows also that there is a *query detector* in the proxy layer at the client side (this layer receives the tuple space operations called from the applications), which is responsible by the already mentioned identification of functional queries. For example, to implement an extended shared counter, servers must access and update the counter value stored into a tuple field [8], which could be done using an OP field. Moreover, to implement a distributed queue, servers must define the order among the tuples available in the space [8] (tuples represent list entries), which could be done using OR fields. Notice that these extended coordination services cannot be implemented with security properties without the extensions proposed in this paper since they allow these computations at the servers.

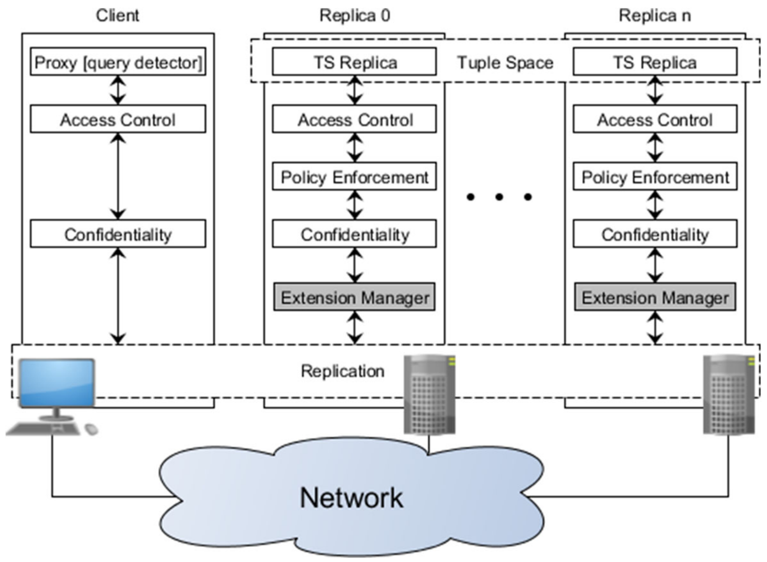

Fig. 4 DepSpace layers with the extension manager

6 Experimental evaluation

In order to assess the performance of the proposed solutions and better understand the costs to provide privacy in the tuple space model, we conducted some experiments with the previously described implementation in Emulab [33].

6.1 Experimental setup

The Emulab environment was configured with 5 d710 machines (2.4 GHz 64-bit Intel Quad Core Xeon E5530 with 2 CPU threads per core, 12 GB of RAM and 1 Gbps network cards) and a 1Gbps switched network. The software installed on the machines was Ubuntu 14 64-bit and a 64-bit Java virtual machine version 1.8.0_121. For all experiments, the system was configured with four replicas hosted in separate machines to tolerate up to one replica failure, while the clients were executed in the remaining machine.

We employed this library without the composition with a ORE algorithm (see Section 3.2). This library outputs a deterministic BigInteger that preserves exactly the same order than the non encrypted number, with pseudo-random distances between any two encrypted numbers. To perform inequality queries (e.g.: less than or greater than), we implemented at the proxy layer of the client side a detector to identify the presence of these queries and handle them in a way that the tuple space layer at the servers side can understand, i.e, the tuple space layer was modified to perform matches using these queries. For instance, a tuple field containing the number 10 matches a template field containing a query "less than 11". The current supported queries are: *lt* (less than), *le* (less than or equal to), *eq* (equal to), *gt* (greater than), and *ge* (greater than or equal to).

For executing *out* operations, we used tuples with only defined fields for each configuration described above (e.g.: $\langle 1,2,3\rangle$, $\langle 1,2,3,4,5\rangle$ and $\langle 1,2,3,4,5,6\rangle$). Notice these fields were protected according to their configuration (PU, CO, PR, CD, OP or OR). On the other hand, the templates used for *rdp* and *inp* operations were configured with one defined field while the remaining ones were configured as wildcards (e.g.: $\langle 1,*,*\rangle$, $\langle 1,*,*,*,*\rangle$ and $\langle 1,*,*,*,*,*\rangle$), except for the configurations PR and OP in which all fields were configured as wildcards since it is not possible to execute matches in these fields (Section 7).

We evaluated the raw throughput of the system at the servers and the latency perceived at the clients in all configurations. To evaluate latency, we used one client to execute each operation 1000 times and obtained the 90th percentile and the average time discarding the 10% values with greater variance. On the other hand, to execute throughput experiments, we variated the number of clients (from one to ten) and measured the maximum throughput obtained in each configuration. In order to stress the servers, each client preprocessed 1000 requests (most of the cryptographic costs are at the client side) before sending them to the servers, that measured the throughput periodically at each 100 executed requests. Although the DepSpace does tolerates faults, all performance values were obtained in fault-free executions.

6.2 Results

This section reports the results obtained in the experiments. First, Fig. 5a and b present the 90th percentile

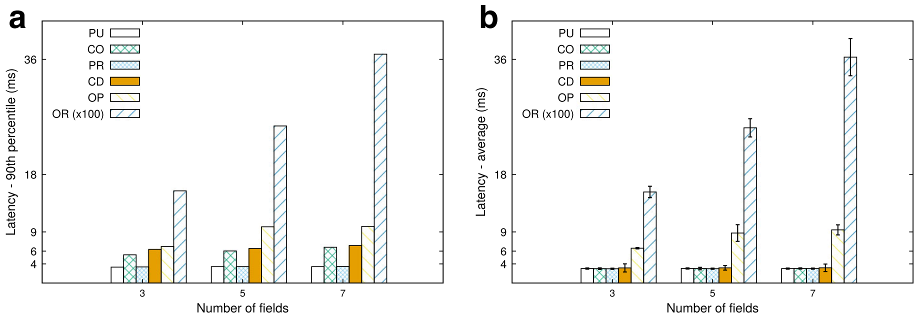

Fig. 5 Tuple insertion (*out*) latency. **a** Latency - 90th Percentile. **b** Latency - Average

and the average latency for the *out* operation, respectively. The cryptographic costs for these operations are related to: *(1)* the execution of the PVSS scheme [5], that is the same cost for all configurations; and *(2)* the generation of the fingerprint, that is different for each approach (see Table 2).

A configuration with fields PU or PR presented the best performance since no cryptographic operations is need to generate the fingerprint, followed by configurations CO and CD. Although more time is necessary to process the fingerprint for CD when compared to CO (Table 2), this fact does not impact its performance since it demands less than a millisecond to execute (0.033 ms). The configuration OP also presented an acceptable performance, although more time is necessary to process the fingerprint. On the other hand, the configuration OR presented poor performance since a lot of time was necessary to process the fingerprint.

Besides that, Table 3 shows the amount of information that must be exchanged among clients and servers for each configuration. The request size is important since it goes through the replication protocols that have complexity of $O(n^2)$ messages [5]. The reply size also is important since every server may send it to the client, resulting in a communications of n to 1.

Table 2 Costs related to cryptographic processing necessary to *(1)* generate a fingerprint, *(2)* verify if a fingerprint is good for a tuple received from servers, *(3)* extract a operable field from the fingerprint and *(4)* execute a match at the servers

	Fingerprint ($\overline{av}$ / σ)	Verify ($\overline{av}$ / σ)	Extract ($\overline{av}$ / σ)	Query/Match ($\overline{av}$ / σ)
PU	–	–	–	–
CO	(0.003 / 0.0003)	(0.003 / 0.0003)	–	–
PR	–	–	–	–
CD	(0.033 / 0.0067)	(0.033 / 0.0067)	–	–
OP	(0.619 / 0.0322)	–	(0.601 / 0.1064)	–
OR	(375.091 / 2.3129)	(375.348 / 1.9041)	–	(0.001 / 0.0002)

All values presented in the table consider the costs to process only one field and it is presented the average ($\overline{av}$) and the standard deviation (σ) in milliseconds (ms)

Figure 6a and b present the latency results for the *rdp* operation. These results are very similar to the results for the *inp* operation (Fig. 7a and b), since the only difference is that the *inp* operation removes the tuple from the space. The cryptographic costs for these operations are related to: *(1)* the client must generate the fingerprint $\overline{f}$ for the template; *(2)* servers search for a tuple with a fingerprint that matches $\overline{f}$; *(3)* servers execute the PVSS scheme to extract their shares; and *(4)* the client must execute the PVSS scheme to combine the received shares and recover the tuple and verify if $\overline{f}$ is valid for the received tuple. Moreover, it is necessary to extract the values of operable fields from the fingerprint. Table 2 presents these costs, except for the PVSS scheme that are the same for all configurations. The size of a request/reply also is presented at Table 3 and, as already commented, impacts the communication costs. The performance presented by the system in the execution of reading (*rdp*) and removal (*inp*) operations followed the same pattern of the insert (*out*) operations, basically for the same reasons.

Another very important aspect is that most of the cryptographic costs are executed at the client side, i.e., the costs reported at columns fingerprint, verify and extract of Table 2. Only the query/match (last column of Table 2) for OR fields are executed by the servers during the search for a tuple which has a fingerprint that matches the fingerprint of the template in a read or removal operation. Moreover, most of the costs related to the execution of the PVSS scheme also are placed at the client side [5]. This is important because it shows that it is possible to have a tuple space that ensures the privacy of the information it stores and, at the same time, is scalable. Consequently, although these costs impact the latency, it is not expected to have a significant impact on the system throughput.

Trying to investigate this aspect, Fig. 8 presents the throughput presented at the servers for each operation and configuration. The throughput is similar for all configurations since no significant cryptographic operation is executed at the servers (only a negligible time is demanded to execute a query/march for the OR configuration).

Table 3 Amount of data (in bytes) sent in a request/reply for each configuration and operation

	OUT (request / reply)			RDP/INP (request / reply)		
	3 fields	5 fields	7 fields	3 fields	5 fields	7 fields
PU	(874 / 133)	(917 / 133)	(951 / 133)	(364 / 1081)	(386 / 1122)	(408 / 1156)
CO	(937 / 133)	(1064 / 133)	(1195 / 133)	(336 / 1147)	(358 / 1269)	(380 / 1405)
PR	(794 / 133)	(831 / 133)	(859 / 133)	(288 / 996)	(310 / 1038)	(332 / 1072)
CD	(1144 / 133)	(1399 / 133)	(1665 / 133)	(403 / 1342)	(425 / 1610)	(447 / 1868)
OP	(1763 / 133)	(2433 / 133)	(3113 / 133)	(288 / 1969)	(310 / 2640)	(332 / 3324)
OR	(825 / 133)	(866 / 133)	(916 / 133)	(297 / 1026)	(319 / 1081)	(341 / 1124)

Moreover, the throughput to insert (Fig. 8a) a tuple in the space is higher than the throughput to read (Fig. 8b) or to remove (Fig. 8c) it from the space. In fact, in the execution of a reading or removal operation, servers must extract and verify the shares while no cryptographic function is executed at the servers for insertion operations (Section 2.1).

Another important aspect to be observed in the experiments is that the number of fields in a tuple does not significantly impacted the system performance for both latency and throughput. In fact, for all configurations and operations, the performance for 3, 5, and 7 fields are similar, except for the OR fields since they demanded much more time to generate the fingerprint (Table 2).

7 Discussions

This section presents some important discussions about some aspects of the proposed solutions.

7.1 (Im)possibility of combinations of fields types

The proposed protocol and implementation do not permit that a field assumes more than one type. However, some types could be seen as a combination of some types since they provide equivalent functionalities. Below we present some examples.

- OR type also provides a deterministic encryption and could be used as CD or CO, i.e., it allows equality match queries.
- CO and CD provide the same functionality but with different security levels.
- PU type provides all functionalities, but it is insecure.

On the other hand, the combinations presented below are not possible:

- OP fields are not deterministic since they use an asymmetric randomized algorithm and, therefore, each encryption of the same number with the same key may produce different results. Consequently, it does not provide the functionalities of CD or CO.
- OP is the only secure field that allows computation.
- OR is the only secure field that allows ordering.
- PR type presents the best security level, but does not provide any functionality.

7.2 State machine replication vs. operable fields

Operable fields could be changed through the execution of computations at the servers. These changes occur in the fingerprint instead of in the tuple that is encrypted as a single piece of data. This approach brings two issues that need some attention:

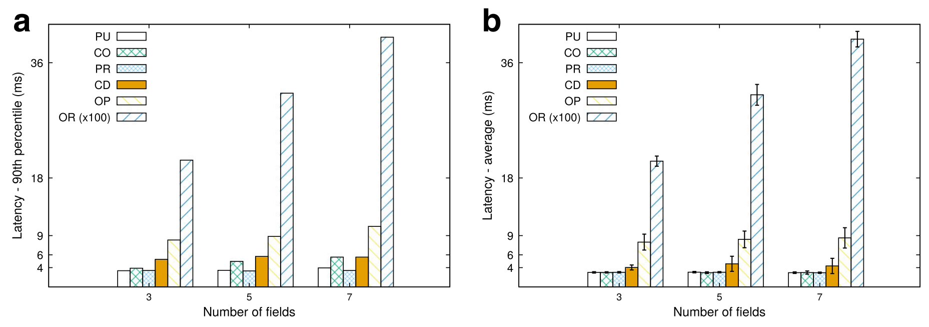

Fig. 6 Tuple reading (*rdp*) latency. **a** Latency - 90th Percentile. **b** Latency - Average

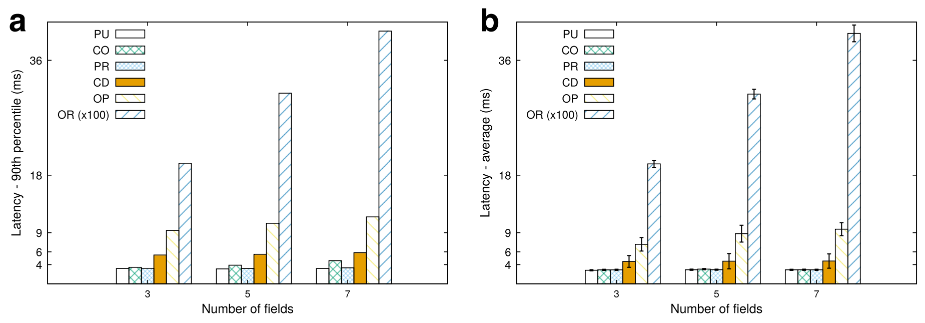

Fig. 7 Tuple removal (*inp*) latency. **a** Latency - 90th Percentile. **b** Latency - Average

- Firstly, when decrypted after a reading/removal operation, the OP fingerprint field may not match the corresponding OP tuple field. Consequently, during this verification (Section 2.1), OP fields are always considered valid. Moreover, the value in the tuple field should be replaced by the updated value in the fingerprint field.
- Secondly, and more critical, during the State Machine Replication (SMR) protocols execution at the client [16, 34], the replies received from the servers may be different since the OP fields use a non-deterministic algorithm. In fact, the cryptograms resulting from some computation at the servers may differ among them, although they refer to the same decrypted value. These aspects impact the SMR protocols, which require $f+1$ identical replies to terminate by returning some of these replies. In order to circumvent this problem, it is necessary to decrypt the OP fields prior to count the number of received replies.

7.3 Key management

Key management almost always is a big challenge in the development of secure systems. Fortunately, in our model we can use the computational power of the tuple space itself for this management. The idea is to use a combination of PU and PR fields to exchange, among the group of communicating processes, the keys used to compute the fingerprint for CD, OP and OR fields. The protocol is quite simple:

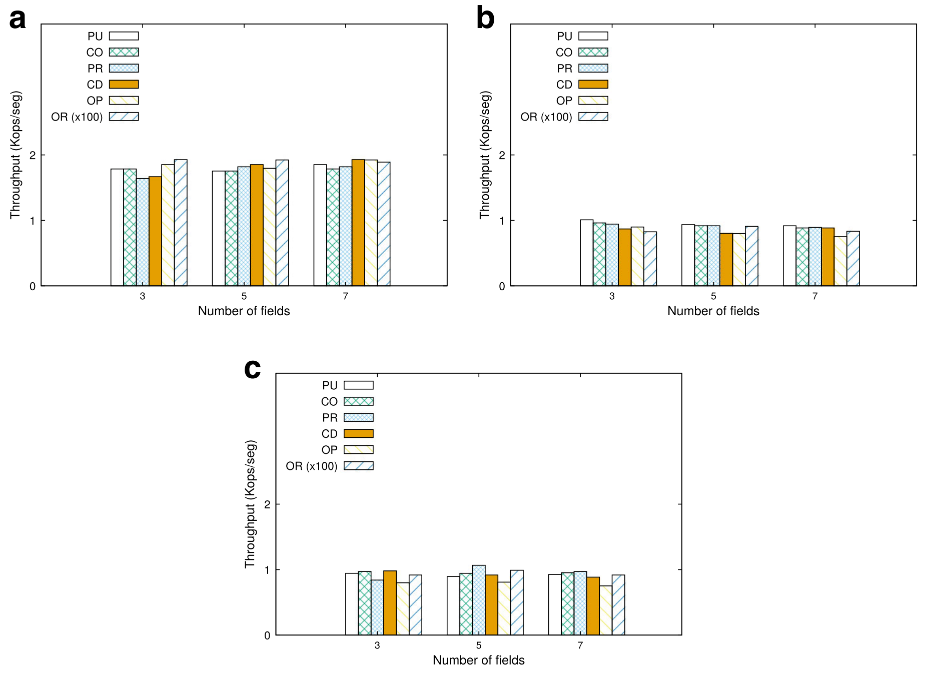

Fig. 8 Throughput for insert (*out*), read (*rdp*) and remove (*inp*) a tuple from the space. **a** Tuple insertion (*out*) throughput. **b** Tuple reading (*rdp*) throughput. **c** Tuple removal (*inp*) throughput

- Each client c generates a secret key k and defines a tuple $t = \langle secret\_key, k\rangle$ and a template $\bar{t} = \langle secret\_key, *\rangle$, both defined as ⟨ PU, PR ⟩ (first field is public and the second is private).
- Later, c invokes a $cas(\bar{t}, t)$ operation. If it returns *null*, then c uses k as a shared secret key since no other previous defined secret key was inserted into the space. Otherwise, the operation returns a tuple $t' = \langle secret\_key, k'\rangle$ read from the space and c uses k' as the shared secret key.

The protocol to exchange a key pair (OP fields) is very similar, the only difference is that the client should create a tuple $t = \langle OP\_keys, k_{pub}, k_{priv}\rangle$ and a template $\bar{t} = \langle OP\_keys, *, *\rangle$ in the first step, both defined as ⟨ PU, PU, PR ⟩.

Finally, the DepSpace access control subsystem must be used and it is enough to ensure that only the group of communicating processes is allowed to access the tuples storing the keys (notice that is is possible to use different keys for different fields). To remove some process from the group, the key must be redefined and shared with the appropriated access control configuration, as previously described. Notice that all tuples/fingerprints generated through the old key need to be replaced by another that uses the new key. On the other hand, the addition of some process in the group is straightforward since a new tuple with the same key but with appropriated access control configuration (including the new process) must be inserted into the space and the joining process only needs to read it from the space.

7.4 Alternatives for fingerprint implementation

Section 5 reports on our choices to implement the fingerprint for each field. However, other alternatives are possible and could be investigated to increase the system security level and/or to boost the system performance. Among many ciphers available in the literature, we discuss below some alternatives for each one of these aspects.

- Increasing the system security level.
  - CD fields: it is possible to improve the security level provided in or implementation for CD fields to achieve the level reported in Table 1 in the following way. First, use the tuple field as input of a HMAC-SHA256 function under the first 16 bytes of a secret key of 256 bits, generating the IV (initialization vector). After, apply the native Java AES CBC mode encryption over the tuple field using the second half of the secret key and the computed IV.
  - OR fields: we used a OPE algorithm alone for OR fields, but it is possible to use it in conjunction with an ORE algorithm to meet the security level reported in Table 1.
- Improving system performance: our implementation for OR fields presented a poor performance because it is based on hyper-geometric distribution functions, which present high computational costs. This is a natural candidate to be replaced by another cipher with the same functionality.

8 Related works

There are several systems developed to provide security and/or fault-tolerance in the tuple spaces model. Among these proposals, the *DepSpace* [5] and its extended version to distributed coordination [8] are the unique to consider both fault-tolerance by using replication techniques and security by combining cryptographic and access control mechanisms. Some of the proposed systems implement only replication mechanisms [12, 38], while others use only access control techniques [9–11]. These proposals for security in the tuple space model have a very limited focus since they consider only simple attacks like invalid access or use weak cryptography mechanisms, which are not enough to ensure the security of the stored information. Other systems provide some level of fault-tolerance by using the concept of transactions [35–37]. Table 4 compares the features provided by these tuple space approaches and the solutions proposed in this paper.

Stronger and robust cryptographic mechanisms were used in the database context aiming to provide confidentiality and protection against information leakage through the processing of queries over encrypted databases [21, 22]. In these works, different cryptography schemes are used in the same application according to the specificity of the data to be stored and the expected queries to be executed over these data (e.g.: equality, comparison and words occurrence in texts). Homomorphic encrypted data, for instance, can accept UPDATE operations without been decrypted. In the same way, OPE and ORE ciphers are used to perform comparison queries, like $\leq$ or $\geq$, with a good performance/security trade-off. Our proposal for new field classification employs these cryptographic mechanisms aiming to mitigate the security problems for applications that use the tuple space programming paradigm.

In CryptDB [21] and related works, such as the framework for searching encrypted databases [22], the performance decreases with the use of cryptographic functions, being more substantial when increasing either the number of client or database sizes. In the proposed system, the server side is not significantly impacted with the execution of cryptographic functions since the heavy cryptographic processing occurs at the client side (Table 2).

Table 4 Comparison among tuple spaces approaches

	Replication	Transaction	Access control	Confidentiality	Allows operations and searches over encrypted data
TSpaces [35]	$\mathcal{X}$	$\surd$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$
JavaSpaces [36]	$\mathcal{X}$	$\surd$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$
GigaSpaces [37]	$\mathcal{X}$	$\surd$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$
FT-Linda [12]	$\surd$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$
Parallel-Linda [38]	$\surd$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$
SecSpaces [10]	$\mathcal{X}$	$\mathcal{X}$	$\surd$	$\mathcal{X}$	$\mathcal{X}$
KLAIM [9]	$\mathcal{X}$	$\mathcal{X}$	$\surd$	$\mathcal{X}$	$\mathcal{X}$
SECOS [11]	$\mathcal{X}$	$\mathcal{X}$	$\surd$	$\mathcal{X}$	$\mathcal{X}$
DepSpace [5]	$\surd$	$\mathcal{X}$	$\surd$	$\surd$	$\mathcal{X}$
DepSpace+this paper solutions	$\surd$	$\mathcal{X}$	$\surd$	$\surd$	$\surd$

9 Conclusions and future work

This paper reports on our efforts to provide privacy in the tuple spaces model by applying robust cryptographic schemes. The main challenge in this model is that tuples are accessed by their contents (associative memory), being necessary to supply some information about them to allow servers to select tuples that match templates. To overcome this problem, this paper proposes the use of robust cryptographic schemes that allow both computations and definition of an order over encrypted data. A set of experiments illustrated the costs related to the proposed solutions.

As future work we intend to investigate the performance of other robust cryptographic libraries that provide the same security level and characteristics of these used in the first implementation. Additionally, we intend to develop a application that uses all the computation power provided by our solution and some secure extensible distributed coordination services, like shared counters, distributed queues and distributed barriers.

Acknowledgements
We would like to thank the universities involved for the support provided. We also thank the many contributions from the reviewers.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Authors' contributions
EF is a student at PPGInf/CIC/UnB and has conducted the analysis about the security of DepSpace and proposed the new fields classifications and implementations. Prof. EA is the advisor of EF and helped mainly in the definitions of the proposals and in the implementations and experiments. Prof. DA is co-advisor of EF and, together with Prof. PS, worked mainly in the security aspects of the work. Moreover, all authors participated in all discussions and in the written phase of the final text. All authors read and approved the final manuscript.

Ethics approval and consent to participate
Not applicable.

Consent for publication
All authors agree to the submitted version.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Department of Computer Science, University of Brasilia, UnB, Brasília, DF, Brazil. [2]Institute of Computing, University of Campinas, UNICAMP, Campinas, SP, Brazil.

Received: 6 September 2017 Accepted: 22 November 2017
Published online: 20 December 2017

References
1. Avizienis A, Laprie JC, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Secure Comput. 2004;1(1):11–33.
2. Veríssimo P. Dialogue on Cyber Policies between Brazil and the EU: prospecting threats and opportunities of the cyberspace. Dialogue Cyber Policies. 2016.
3. Naveed M, Kamara S, Wright CV. Inference attacks on property-preserving encrypted databases. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York: ACM; 2015. p. 644–55.
4. Gelernter D. Generative Communication in Linda. ACM Trans Programing Lang Syst. 1985;7(1):80–112.
5. Bessani AN, Alchieri EP, Correia M, da Silva Fraga J. DEPSPACE: A byzantine fault-tolerant coordination service. In: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008. New York: ACM; 2008. p. 163–76.
6. Alchieri EAP, Bessani AN, Fraga dSJ. A dependable infrastructure for cooperative web services coordination. In: IEEE International Conference on Web Services. Beijing: IEEE; 2008. p. 21–8.
7. Bessani AN, Correia M, Fraga JS, Lung LC. Sharing memory between Byzantine processes using policy-enforced tuple spaces. In: Proceedings of 26th IEEE International Conference on Distributed Computing Systems. Lisboa: IEEE; 2006.
8. Distler T, Bahn C, Bessani A, Fischer F, Junqueira F. Extensible distributed coordination. In: Proceedings of the Tenth European Conference on Computer Systems. EuroSys '15. New York: ACM; 2015.
9. De Nicola R, Ferrari GL, Pugliese R. KLAIM: A Kernel Language for Agents Interaction and Mobility. IEEE Trans Softw Eng. 1998;24(5):315–30.
10. Busi N, Gorrieri R, Lucchi R, Zavattaro G. SecSpaces: a Data-Driven Coordination Model for Environments Open to Untrusted Agents. Electron Notes Theor Comput Sci. 2003;68(3):310–27.

11. Vitek J, Bryce C, Oriol M. Coordinating processes with Secure Spaces. Sci Comput Program. 2003;46(1–2):163–93.
12. Bakken DE, Schlichting RD. Supporting Fault-Tolerant Parallel Programing in Linda. IEEE Trans Parallel Distrib Syst. 1995;6(3):287–302.
13. Segall EJ. Resilient distributed objects: Basic results and applications to shared spaces. In: Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing - SPDP'95. San Antonio: IEEE; 1995. p. 320–7.
14. Lamport L, Shostak R, Pease M. The Byzantine generals problem. ACM Trans Program Lang Syst. 1982;4(3):382–401.
15. Schoenmakers B. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology - CRYPTO'99. Santa Barbara: Springer Berlin Heidelberg. 1999. p. 148–64.
16. Castro M, Liskov B. Practical Byzantine fault-tolerance and proactive recovery. ACM Trans Comput Syst. 2002;20(4):398–461.
17. Menezes AJ, Vanstone SA, Oorschot PCV. Handbook of Applied Cryptography, 1st. Boca Raton: CRC Press, Inc.; 1996.
18. Boneh D, Shoup V. A Graduate Course in Applied Cryptography. 2015. https://crypto.stanford.edu/~dabo/cryptobook/draft_0_2.pdf. Accessed Nov 2017.
19. Bellare M, Desai A, Pointcheval D, Rogaway P. Relations among notions of security for public-key encryption schemes. In: Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference. Santa Barbara: Springer Berlin Heidelberg. 1998. p. 26–45.
20. Boldyreva A, Chenette N, Lee Y, O'Neill A. Order-Preserving Symmetric Encryption. 2012. Cryptology ePrint Archive, Report 2012/624. http://eprint.iacr.org/2012/624. Accessed Nov 2017.
21. Popa RA, Redfield CMS, Zeldovich N, Balakrishnan H. CryptDB: Protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. New York: ACM; 2011. p. 85–100.
22. Alves PGMR, Aranha DF. A framework for searching encrypted databases. In: XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSEG 2016). Niterói: SBC; 2016. p. 142–55.
23. Boneh D, Lewi K, Raykova M, Sahai A, Zhandry M, Zimmerman J. Semantically Secure Order-Revealing Encryption: Multi-Input Functional Encryption Without Obfuscation. 2014. Cryptology ePrint Archive, Report 2014/834. http://eprint.iacr.org/2014/834. Accessed Nov 2017.
24. Chenette N, Lewi K, Weis SA, Wu DJ. Practical Order-Revealing Encryption with Limited Leakage. 2015. Cryptology ePrint Archive, Report 2015/1125. http://eprint.iacr.org/2015/1125. Accessed Nov 2017.
25. Lewi K, Wu DJ. Order-revealing encryption: New constructions, applications, and lower bounds. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16. New York: ACM; 2016. p. 1167–78.
26. Tourky D, ElKawkagy M, Keshk A. Homomorphic encryption the "holy grail" of cryptography. In: 2016 2nd IEEE International Conference on Computer and Communications (ICCC). Chengdu: IEEE; 2016. p. 196–201.
27. Naehrig M, Lauter K, Vaikuntanathan V. Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop. New York: ACM; 2011. p. 113–24.
28. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In: Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques. EUROCRYPT'99. Berlin: Springer-Verlag; 1999. p. 223–38.
29. Gamal TE. A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theory. 1985;31(4):469–72.
30. Khader AS, Lai D. Preventing man-in-the-middle attack in diffie-hellman key exchange protocol. In: 22nd International Conference on Telecommunications. Sydney: IEEE; 2015. p. 204–8.
31. Analytics N. A Java library for Paillier partially homomorphic encryption. Available at https://github.com/n1analytics/javallier. Accessed Nov 2017.
32. Savvides S. Order-preserving encryption in Java. Available at https://github.com/ssavvides/jope. Accessed Nov 2017.
33. White B, Lepreau J, Stoller L, Ricci R, Guruprasad S, Newbold M, Hibler M, Barb C, Joglekar A. An Integrated Experimental Environment for Distributed Systems and Networks. In: Proc. of 5th Symp. on Operating Systems Design and Implementations. Boston: ACM; 2002.
34. Schneider FB. Implementing fault-tolerant service using the state machine aproach: A tutorial. ACM Comput Surv. 1990;22(4):299–319.
35. Lehman TJ, et al. Hitting the distributed computing sweet spot with TSpaces. Comput Netw. 2001;35(4):457–72.
36. JavaSpaces. JavaSpaces Guide. 2016. Available at http://www.oracle.com/technetwork/articles/java/javaspaces-140665.html. Accessed Nov 2017.
37. GigaSpaces. GigaSpaces Homepage. 2016. Available at http://www.gigaspaces.com/. Accessed Nov 2017.
38. Xu A, Liskov B. A design for a fault-tolerant, distributed implementation of Linda. In: Proc. of the 19th Symposium on Fault-Tolerant Computing. Chicago: IEEE; 1989. p. 199–206.