# Privacidade em Dados Armazenados em Memória Compartilhada através de Espaços de Tuplas

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

[1] Departamento de Ciência da Computação – Universidade de Brasília – UnB
[2]Instituto de Computação – Universidade Estadual de Campinas – UNICAMP

***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 them susceptible to attacks that impair user and data privacy. To overcome this limitation, this work proposes some extensions to* DEPSPACE*, a tuple space that implements dependability properties, to improve data privacy. A deep analysis concerning security aspects of this system is presented, as well as the benefits of the proposed solutions.*

***Resumo.*** *Um espaço de tuplas pode ser visto conceitualmente como um objeto de memória compartilhada que fornece operações para armazenar e recuperar conjuntos de dados ordenados chamados tuplas. As tuplas armazenadas em um espaço de tuplas são acessadas através do conteúdo de seus campos, funcionando assim como uma memória associativa. Embora existam algumas propostas que visam adicionar propriedades de segurança em espaços de tuplas, a necessidade de acesso através do conteúdo dos campos faz com que as mesmas sejam susceptíveis a ataques capazes de quebrar a privacidade dos dados e usuários do espaço. Visando contornar estes problemas, este trabalho propõe extensões ao* DEPSPACE*, um espaço de tuplas com propriedades de segurança, com o intuito de impedir que a privacidade dos dados e usuários seja afetada. Uma vasta análise acerca da segurança deste sistema é apresentada, bem como dos ganhos advindos com as soluções propostas.*

## 1. Introdução

A preocupação com aspectos de segurança tem ganhado espaço cada vez maior no projeto e desenvolvimento de aplicações distribuídas. Um sistema é dito seguro se satisfaz requisitos de integridade, disponibilidade e confidencialidade [Avizienis et al. 2004]. Intuitivamente, privacidade é entendida na perspectiva de uma entidade como a confidencialidade de suas informações sensíveis (dados e metadados) [Veríssimo 2016]. Esta entidade pode ser uma pessoa, uma organização, uma nação, etc. Como podemos observar, a privacidade está diretamente relacionada com a confidencialidade das informações.

Atualmente existem vários fatores que aumentam o risco à segurança das aplicações [Veríssimo 2016]: *(i)* o mundo está se tornando uma infraestrutura imensa, interconectada e interdependente; *(ii)* existem muitos dados correlacionados disponíveis; *(iii)* as entidades estão se expondo cada vez mais; e *(iv)* o número de vulnerabilidades de *software* está aumentando.

Em meio a este cenário, muitos sistemas visam manter a confidencialidade protegendo apenas os dados sigilosos, sem se preocupar com os dados não sigilosos relacionados. Entretanto, tendo em vista que ataques de inferência estatística, através da análise e correlação de dados de acesso público, muitas vezes conseguem obter informações mantidas sob sigilo [Naveed et al. 2015], torna-se interessante, sob o viés de segurança, proteger toda informação disponibilizada para uma aplicação.

Estes aspectos são particularmente relevantes quando consideramos dados compartilhados através de espaços de tuplas [Gelernter 1985, Bessani et al. 2008], onde o acesso é concretizado através do conteúdo dos campos das tuplas (memória associativa). Embora existam algumas propostas que visam adicionar propriedades de segurança neste modelo [De Nicola et al. 1998, Busi et al. 2003, Vitek et al. 2003, Bessani et al. 2008, Distler et al. 2015], a necessidade de acesso através do conteúdo dos campos faz com que as mesmas sejam susceptíveis a ataques capazes de quebrar a privacidade dos dados e usuários do espaço. Dentre as propostas existentes, o DEPSPACE [Bessani et al. 2008] é o sistema que provê um maior nível de segurança, empregando mecanismos tanto de controle de acesso quanto de criptografia. Este sistema sugere a classificação dos campos das tuplas em público, comparável ou privado. Para que o acesso a uma tupla seja possível, pelo menos um de seus campos precisa ser público ou comparável. Esta limitação torna o sistema susceptível a ataques de inferência estatística, conforme já discutido, ou ainda dificulta a sua utilização por aplicações, visto que as possibilidades de buscas por tuplas podem ser significativamente reduzidas para se preservar a segurança da informação.

Visando contornar estes problemas, este artigo propõe extensões ao DEPSPACE com o intuito de impedir que a privacidade dos dados e usuários seja afetada. Através do emprego de mecanismos de criptografia mais recentes e sofisticados, são preservadas tanto as propriedades de segurança do sistema quanto a flexibilidade das buscas no modelo tradicional de coordenação por espaço de tuplas. Desta forma, obtém-se um modelo em que a inclusão de segurança não afeta as possibilidades de buscas por tuplas. Adicionalmente, este trabalho faz uma vasta análise acerca da segurança do DEPSPACE, bem como dos aprimoramentos advindos com as soluções propostas.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 detalha o conceito de espaço de tuplas e apresenta o DEPSPACE, além de fazer uma análise acerca da segurança fornecida por este sistema. Na Seção 3 são discutidos esquemas criptográficos robustos que são utilizados na extensão proposta para o DEPSPACE, apresentada na Seção 4. Finalmente, os trabalhos relacionados relevantes são discutidos na Seção 5, enquanto que as conclusões e os trabalhos futuros são apresentados na Seção 6.

## 2. Espaço de Tuplas

Um espaço de tuplas pode ser visto (conceitualmente) como um objeto de memória compartilhada que fornece operações para armazenar e recuperar conjuntos de dados ordenados chamados de tuplas. Sendo assim, os processos de um sistema distribuído podem interagir através desta abstração de memória compartilhada. Uma tupla $t$ é uma sequência ordenada de campos, onde um campo que contém um valor é dito definido. Um tupla onde todos os campos são definidos é chamada de entrada. Uma tupla $\overline{t}$ é chamada molde (ou *template*) se algum de seus campos não tem valor definido. Diz-se que uma tupla $t$ e um molde $\overline{t}$ combinam se e somente se ambos têm o mesmo número de campos e todos os valores e tipos dos campos definidos em $\overline{t}$ são iguais aos valores e tipos dos campos cor-

respondentes em $t$. Por exemplo, uma tupla $\langle \text{SBRC}, 2017, \text{Belém} \rangle$ combina com o molde $\langle \text{SBRC}, 2017, * \rangle$ ('$*$' denota um campo sem definição do molde).

A coordenação por espaço de tuplas (generativa), introduzida pela linguagem de programação para sistemas paralelos LINDA [Gelernter 1985], suporta comunicações desacopladas no espaço (os processos não precisam conhecer as localizações uns dos outros) e no tempo (os processos não precisam estar ativos ao mesmo tempo). Além disso, este modelo de coordenação fornece algum poder de sincronização entre processos.

As manipulações realizadas no espaço de tuplas consistem em invocações de três operações básicas [Gelernter 1985]: $out(t)$ que adiciona a entrada $t$ no espaço de tuplas; $in(\bar{t})$, que remove do espaço de tuplas uma tupla que combina com o molde $\bar{t}$; $rd(\bar{t})$, usada na leitura de uma tupla que combina com o molde $\bar{t}$, sem removê-la do espaço. As operações $in$ e $rd$ são bloqueantes, i.e., se não houver uma tupla que combine com o molde no espaço, o processo fica bloqueado até que uma esteja disponível. Uma extensão comum a este modelo é a inclusão de variantes não bloqueantes das operações de leitura, denominadas $inp$ e $rdp$. Estas operações funcionam exatamente como as anteriores, a não ser pelo fato de retornarem mesmo não havendo uma tupla que combine com o molde usado (indicando esta inexistência). Note que, de acordo com as definições anteriores, o espaço de tuplas funciona como uma memória associativa: os dados são acessados a partir de seu conteúdo, e não através de seu endereço.

### 2.1. DEPSPACE: Um Sistema de Coordenação Tolerante a Falhas Bizantinas

Segurança é uma característica fundamental de sistemas confiáveis [Avizienis et al. 2004]. No contexto de um espaço de tuplas, os seguintes atributos são necessários: *confiabilidade* (as operações realizadas no espaço de tuplas fazem com que seu estado se modifique de acordo com a especificação), *disponibilidade* (o espaço de tuplas sempre está pronto para executar as operações requisitadas por partes autorizadas), *integridade* (nenhuma alteração imprópria no estado de um espaço de tuplas pode ocorrer), *confidencialidade* (o conteúdo dos campos das tuplas não pode ser revelado a partes não autorizadas). O DEPSPACE [Bessani et al. 2008] é a implementação de um espaço de tuplas que busca satisfazer estas propriedades por meio de diversas camadas, cada uma responsável pela concretização de uma funcionalidade diferente.

#### 2.1.1. Camadas do DEPSPACE

Esta seção apresenta as camadas do DEPSPACE [Bessani et al. 2008], dando ênfase à camada de confidencialidade, que é responsável pelos aspectos discutidos neste trabalho.

**Replicação.** Para manter a consistência do espaço de tuplas, o DEPSPACE utiliza replicação Máquina de Estados [Schneider 1990, Castro and Liskov 2002]. Este mecanismo está relacionado principalmente com as propriedades de disponibilidade e confiabilidade, pois para o número $n$ de réplicas no sistema, garante que o espaço de tuplas executa as operações a ele endereçadas seguindo sua especificação, mesmo que até $f = (n-1)/3$ réplicas sejam maliciosas (as réplicas corretas *mascaram* o comportamento das maliciosas). Através deste protocolo, as réplicas corretas executam a mesma sequência de operações e retornam os mesmos valores, evoluindo de forma sincronizada.

**Confidencialidade.** Conforme comentado, as tuplas são mantidas replicadas em um conjunto de servidores no DEPSPACE. Sendo assim, a preservação da propriedade de confidencialidade não pode ser atribuída a um único servidor, pois até $f$ deles podem falhar e revelar o conteúdo das tuplas a partes não autorizadas.

Desta forma, a confidencialidade é implementada através do uso de um ($n$,$f + 1$) – esquema de compartilhamento de segredo publicamente verificável (*publicly verifiable secret sharing* – PVSS) [Schoenmakers 1999]. Os clientes, que são os distribuidores deste esquema, cifram as tuplas com um segredo por eles gerado. Após isso, geram um conjunto de $n$ fragmentos (*shares*) deste segredo. Um segredo pode ser remontado apenas com a combinação de $f + 1$ *shares*, o que torna impossível que um conluio de até $f$ servidores faltosos revele o conteúdo de uma tupla. Este esquema utiliza um *fingerprint* da tupla para implementar a comparação entre tuplas e moldes, o qual é computado de acordo com o tipo dos campos escolhidos para a tupla, os quais podem ser:

- **Público (PU)**, o próprio valor do campo é o *fingerprint*, i.e, nenhum método criptográfico é aplicado ao conteúdo do campo que fica exposto;
- **Comparável (CO)**, um *hash* do valor do campo é o *fingerprint* (para isso utiliza uma função de *hash* resistente a colisões), possibilitando a execução de buscas (comparações); e
- **Privado (PR)**, um símbolo especial é o *fingerprint*, i.e., o conteúdo deste campo é mantido cifrado sem qualquer possibilidade de realização de comparações ou acesso aos dados.

Como não é possível enviar diferentes versões de uma requisição para diferentes servidores (contendo apenas seu *share* do segredo usado para cifrar a tupla), o cliente deve cifrar cada um dos *shares* com uma chave secreta compartilhada com o servidor que vai armazenar esse *share*. Deste modo, cada servidor terá acesso apenas ao seu *share* (um servidor faltoso não têm acesso a todos os *shares* para remontar e revelar a tupla). Assim, nas requisições de inserção de tuplas, o cliente envia aos servidores a tupla cifrada, os *shares* cifrados, as provas de que estes *shares* são válidos e o *fingerprint* da tupla.

Para acessar uma tupla, o cliente envia o *fingerprint* do molde e espera pelas respostas dos servidores. A resposta de cada servidor contém o *fingerprint* da tupla (que combina com o *fingerprint* do molde), a tupla cifrada e o *share* armazenado por este servidor[1] (juntamente com a prova de sua validade). O cliente decifra os *shares*, verifica suas validades e combina $(f + 1)$ deles para obter o segredo e decifrar a tupla. Note que um cliente malicioso pode inserir uma tupla e informar um *fingerprint* que não corresponde ao *fingerprint* da tupla. Deste modo, após obter a tupla, o cliente deve verificar se a tupla corresponde ao *fingerprint*. Caso isso não aconteça, o cliente precisa eliminar esta tupla do espaço (se ainda não eliminou) e re-executar a operação. A eliminação de tuplas inválidas é realizada em dois passos: *(1)* o cliente envia todas as respostas recebidas para os servidores como prova que esta tupla é inválida (para isso, os servidores devem assinar as respostas a estas requisições); e *(2)* os servidores verificam a autenticidade das respostas e, se a tupla realmente é inválida, removem-na de seus espaços locais.

Vale destacar que pela forma como o *fingerprint* da tupla é definido, é necessário que cada tupla tenha campos públicos e/ou comparáveis de acordo com as buscas que

[1]O *share* é cifrado com a chave secreta compartilhada com o cliente para evitar captura das respostas.

serão realizadas (moldes utilizados nas buscas), sendo impossível suportar uma tupla apenas com campos privados pois nenhuma busca seria possível, i.e., os campos privados não podem ser utilizados para verificar se uma tupla e um molde combinam e são sempre usados como campos indefinidos no molde. Esta limitação traz pelo menos duas consequências. Por um lado, uma tupla com muitos campos privados faz com que a busca fique muito restrita e sem refinamento adequado, perdendo-se a flexibilidade na escolha de moldes, pois um molde com muitos campos indefinidos impossibilita uma busca mais refinada e limita o seu uso pelas aplicações. Por outro lado, uma tupla com muitos campos (dados) públicos e/ou comparáveis está susceptível a ataques, como veremos adiante.

**Controle de Acesso** O controle de acesso é um mecanismo fundamental para manutenção da integridade e confidencialidade das informações (tuplas) armazenadas no DEPSPACE, pois previne que clientes não autorizados obtenham acesso as tuplas, além de impedir que clientes faltosos saturem o espaço de tuplas enviando uma grande quantidade de tuplas. Atualmente, o DEPSPACE implementa controle de acesso de duas formas:

- **Baseado em credenciais:** para cada tupla inserida no DEPSPACE pode-se definir quais são as credenciais necessárias para acessá-la, tanto para leitura quanto para remoção (acesso em nível de tuplas). Estas credenciais são definidas pelo processo que insere a tupla. Também é possível definir, quando o espaço de tuplas é criado, quais são as credenciais necessárias para inserir uma tupla no espaço (acesso em nível de espaço). A implementação desta funcionalidade é realizada através da associação de listas de controle de acesso a cada espaço e tupla.
- **Políticas de granularidade fina:** o DEPSPACE suporta a definição de políticas de acesso de granularidade fina [Bessani et al. 2006], que devem ser especificadas no momento da criação do espaço de tuplas. Estas políticas controlam o acesso ao espaço considerando três parâmetros: o identificador do cliente, a operação que será executada (juntamente com seus argumentos) e o estado do espaço.

### 2.1.2. Análise de Segurança

Apresentaremos a seguir uma rápida explanação sobre algumas definições de segurança. Segundo [Menezes et al. 1996], os ataques aos esquemas criptográficos buscam obter o texto claro ou a chave de decifração através dos seguintes métodos:

- *Ciphertext-only attack (COA)*: Neste tipo de ataque, um adversário pode obter a chave de decifração ou o texto claro somente de posse do texto cifrado. Este é o tipo de ataque mais fraco e, portanto, um sistema vulnerável a este tipo de ataque não tem qualquer tipo de segurança.
- *Know-plaintext attack (KPA)* ou Ataque de texto claro conhecido: Neste tipo de ataque o adversário tem ao seu dispor uma significativa quantidade de textos claros e seus correspondentes textos cifrados e através desta comparação tenta obter a chave de decifração ou decifrar outros textos cifrados.
- *Chosen-plaintext attack (CPA)* ou ataque de texto claro escolhido: Neste tipo de ataque o adversário escolhe um texto claro e lhe é fornecido o texto cifrado correspondente, ele usa então a análise desta correlação para obter o texto claro correspondente a outro texto cifrado.

- *Adaptative chosen-plaintext attack (CPA2)*: Semelhante ao anterior, porém o atacante pode escolher novos textos claros dependendo da resposta recebida.
- *Chosen ciphertext attack (CCA)* ou ataque de texto cifrado escolhido: Neste tipo de ataque, o adversário escolhe um texto cifrado e lhe é fornecido (sem acesso à chave de decifração) o texto claro correspondente, ele então usa a análise desta correlação para obter o texto claro correspondente a outro texto cifrado.
- *Adaptative chosen-ciphertext attack (CCA2)*: Semelhante ao anterior, porém o atacante pode escolher novos textos cifrados dependendo da resposta recebida. Este ataque é considerado muito forte e de implementação mais difícil.

Os ataques citados estão em ordem de complexidade, de modo que um sistema vulnerável a um ataque mais fraco será classificado em um nível de segurança inferior, mesmo que resista a um ataque mais forte. Apesar de serem estes os principais ataques considerados pela literatura, uma infinidade de outros ataques pode ser possível dependendo das características do sistema. Por exemplo, [Naveed et al. 2015] mostram que é possível realizar os chamados *Ataques de inferência* por meio da correlação dos textos cifrados com informações adicionais publicamente disponíveis. Neste caso, havendo uma correlação forte entre estes dados cifrados e os públicos, os textos claros podem ser recuperados com grande acurácia. Em seu trabalho com bases de dados cifradas de hospitais, mais de 60% dos dados cifrados deterministicamente (Seção 3.1), como sexo, raça e risco de mortalidade, foram descobertos em 60% dos hospitais, enquanto que mais de 80% dos dados cifrados com preservação de ordem (Seção 3.2), como idade e nível de severidade de doença, foram recuperados em 95% dos hospitais.

Como na prática é impossível alcançar total segurança contra estes ataques para todos os adversários *matematicamente* possíveis, faz-se necessário um enfraquecimento da definição de segurança, levando em consideração apenas os adversários *computacionalmente* possíveis. Sob essa ótica, define-se informalmente como *semanticamente seguro* um sistema que, com alta probabilidade, é capaz de resistir aos ataques realizados por qualquer adversário computacionalmente eficiente [Boneh and Shoup 2015]. Baseados nas definições formais de [Bellare et al. 1998], definimos informalmente que para todo adversário eficiente **A**, uma cifra $\mathcal{E} = (E, D)$ definida sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ pode oferecer:

- **Indistinguibilidade contra *ataques de texto claro escolhido (IND-CPA)*** se, para qualquer tentativa $i = 1, 2, ...q$, dadas duas mensagens $m_{i0}, m_{i1} \in \mathcal{M}$ de mesmo tamanho escolhidas por **A** e submetidas a um oráculo que responde com a cifra $c_i = E(k, m_{ib}) \in \mathcal{C}$ para alguma chave $k$ selecionada aleatoriamente em $\mathcal{K}$ e $b \in \{0, 1\}$, a probabilidade de que **A** possa distinguir se $c_i = E(k, m_{i0})$ ou $c_i = E(k, m_{i1})$ é desprezível.
- ***Indistinguibilidade contra ataques de texto cifrado escolhido (IND-CCA)*** se, para as mesmas condições do item IND-CPA, o adversário $\mathcal{A}$ ainda obtiver acesso a um oráculo que dado um texto cifrado $c_i \notin \{c_1, ..., c_{i-1}\}$ responde com o **texto claro** $m_i = D(k, c_i)$ correspondente e da mesma forma a probabilidade de que $\mathcal{A}$ possa distinguir se $c_i = E(k, m_{i0})$ ou $c_i = E(k, m_{i1})$ é desprezível. Neste caso, $\mathcal{A}$ pode fazer quantas requisições quiser ao oráculo de decifração, porém somente até receber o criptograma desafio do oráculo de cifração.
- ***Indistinguibilidade contra ataques adaptativos de texto cifrado escolhido (IND-CCA2)*** se, além do estabelecido no item IND-CCA, o adversário puder continuar a usar o oráculo de decifração mesmo após receber o criptograma desafio, tendo como única restrição não poder submeter este criptograma para decifração.

Adicionalmente, temos as seguintes flexibilizações de IND-CPA para cifras determinísticas e de ordem:

- **Indistinguibilidade contra *Ataque de texto claro distinto escolhido (IND-DCPA)*** se a cifra $\mathcal{E}$ for determinística e se, para qualquer tentativa $i = 1, 2, ..., q$, dadas duas mensagens $m_{i0}, m_{i1} \in \mathcal{M}$ de mesmo tamanho escolhidas por **A**, distintas para cada tentativa, isto é, $\forall i, j \in \{1, 2, ..., q\}, m_{i0} \neq m_{j0}$ e $m_{i1} \neq m_{j1}$, submetidas a um oráculo que responde com a cifra $c_i = E(k, m_{ib}) \in \mathcal{C}$ para alguma chave $k$ selecionada aleatoriamente em $\mathcal{K}$ e $b \in \{0, 1\}$, a probabilidade de que **A** possa distinguir se $c_i = E(k, m_{i0})$ ou $c_i = E(k, m_{i1})$ é desprezível [Bellare et al. 2004].
- **Indistinguibilidade contra *Ataques de texto claro ordenado escolhido (IND-OCPA)*** se a cifra $\mathcal{E}$ preservar a ordem dos textos claros e se, para qualquer tentativa $i = 1, 2, ...q$, dadas duas mensagens $m_{i0}, m_{i1} \in \mathcal{M}$ de mesmo tamanho escolhidas por **A** e submetidas sempre na mesma ordem (isto é, $m_{i0} < m_{j0} \iff m_{i1} < m_{j1}$ para todo $1 \leq i, j \leq q$) a um oráculo que responde com a cifra $c_i = E(k, m_{ib}) \in \mathcal{C}$ para alguma chave $k$ selecionada aleatoriamente em $\mathcal{K}$ e $b \in \{0, 1\}$, a probabilidade de que **A** possa distinguir se $c_i = E(k, m_{i0})$ ou $c_i = E(k, m_{i1})$ é desprezível [Boldyreva et al. 2012].

Com isso podemos perceber que o sistema DEPSPACE apresenta algumas vulnerabilidades que devem ser pontuadas. Os campos comparáveis, apesar de trazerem ao sistema a grande vantagem de permitir a seleção de tuplas sem a necessidade de decifrá-las, por utilizar funções de *hash* são vulneráveis a ataques de pré-imagem, pois um adversário sempre pode obter qualquer quantidade desejada de entradas e suas respectivas saídas simplesmente calculando seus *hashes*. Este ataque assemelha-se ao ataque de texto claro conhecido, exceto pelo fato de que neste caso não há uma função de cifração ou decifração. Desta forma, se o conjunto de valores que um campo como este pode assumir for pequeno e conhecido, o atacante pode calcular os *hashes* de todos os valores possíveis, obtendo assim os textos claros correspondentes. Somado a isso, a existência de campos públicos pode proporcionar a um atacante a possibilidade de correlação entre os dados cifrados do sistema e uma base pública favorecendo os ataques de inferência.

Além disso, para prover o uso de criptografia sobre os dados armazenados nas tuplas, [Bessani et al. 2008] propõem deixar partes das chaves de decifração distribuídas entre os servidores, de modo que para recuperar a informação, um cliente deve receber um número mínimo de respostas $(f + 1)$ que possibilitam a composição da chave por completo. Porém, em um ambiente não confiável, um número $q \geq f + 1$ de servidores maliciosos poderiam entrar em conluio, juntar suas partes e recuperar as chaves, conseguindo acesso total ao conteúdo das tuplas, comprometendo toda a segurança do sistema.

## 3. Esquemas Criptográficos mais Robustos

Na busca por solucionar as vulnerabilidades e limitações citadas, apresentamos alguns modelos que permitem realizar buscas ou computações sobre dados cifrados e propomos como podem ser adaptados para o DEPSPACE. Baseados nas características do DEPSPACE, buscamos os esquemas de criptografia que melhor se encaixam em nossas necessidades em meio às muitas propostas existentes na literatura.

### 3.1. Cifras Determinísticas e Probabilísticas

Uma cifra $\mathcal{E} = (E, D)$ definida sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ é dita *probabilística* se para entradas fixas de chave $k \in \mathcal{K}$ e mensagem $m \in \mathcal{M}$ da função de cifração $E : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$,

a saída $c = E(k, m)$ pode assumir valores diferentes. Caso contrário, a cifra será dita *determinística* [Boneh and Shoup 2015].

Cifras determinísticas possuem a característica de vazar igualdade de textos claros quando cifrados sob a mesma chave, ou seja, $m_0 = m_1 \iff E(k, m_0) = E(k, m_1)$. Essa característica pode ser utilizada para realizar buscas por igualdade em dados cifrados, como mostrado em [Popa et al. 2011, Alves and Aranha 2016]. Este tipo de cifra claramente não alcança segurança contra o ataque de texto claro escolhido, visto que um adversário pode enviar inicialmente ao oráculo duas cópias da mesma mensagem $m_0$ e receber portanto $c_0 = E(k, m_0)$, logo em seguida, envia as mensagens $m_0$ e $m_1$, recebendo em resposta $c_b = E(k, m_b)$ com $b \in \{0, 1\}$. Basta então comparar $c_0$ e $c_b$ para ter a certeza de que se trata de $E(k, m_0)$ ou $E(k, m_1)$.

Já as cifras probabilísticas (ou aleatorizadas), possuem a característica de resistirem a ataques mais fortes, podendo atingir o nível de segurança IND-CCA2 quando combinadas com primitivas criptográficas de autenticação.

### 3.2. Order-Preserving Encryption - OPE

Boldyreva [et. al] propõe em [Boldyreva et al. 2012] um esquema criptográfico simétrico que preserva a relação de ordem, ou seja, para todo $i$ e $j$ e para toda chave $k$, $E(k, i) < E(k, j) \iff i < j$. Uma implementação com esta característica permite o vazamento de informações de ordem entre os textos cifrados, não satisfazendo portanto à premissa de IND-CPA. Admitindo que esta premissa não é alcançável por um algoritmo determinístico com tal propriedade, mesmo ante a uma flexibilização da premissa de segurança (IND-OCPA), os autores propõem utilizar funções e geradores pseudo-aleatórios (PRF's e PRG's) e provam que este desfruta de um nível flexibilizado porém ainda forte de segurança por eles chamado de *pseudorandom order-preserving function under chosen-ciphertext attack* (POPF-CCA), ou seja, uma função pseudo-aleatória com preservação de ordem que resiste ao ataque de texto cifrado escolhido.

Entretanto, esta definição de segurança não especifica quais dados, além da ordem, podem vazar ao se utilizar tal cifra. Em [Boneh et al. 2014] é proposta *Order Revealing Encryption (ORE)*, uma construção que minimiza a quantidade de dados vazados e portanto possui um nível mais robusto de segurança, porém inviável em termos de desempenho. Logo depois, [Chenette et al. 2015] propõem um algoritmo prático de ORE que atinge o nível de segurança IND-OCPA, porém com o vazamento do primeiro *bit* que diferencia os valores comparados. Posteriormente, [Lewi and Wu 2016] apresentam um algoritmo de ORE resistente aos ataques de inferência e que funciona muito bem em aplicações de Banco de Dados cifrados, como pode ser visto em [Alves and Aranha 2016].

### 3.3. Criptografia Homomórfica

A Criptografia Homomórfica [Morais and Dahab 2012] tem sido pesquisada como uma forma de se realizar computação em dados cifrados sem a necessidade de decifrá-los ou conhecer a chave de decifração. Isso se deve ao fato de que, em uma cifra homomórfica, dadas duas mensagens quaisquer $m_1$ e $m_2$, uma função de encriptação $E$ parametrizada por uma chave $k$, tem-se que $E(k, m_1) \circ_c E(k, m_2) = E(k, m_1 \circ_m m_2)$, onde $\circ_m$ denota uma operação aritmética no domínio das mensagens e $\circ_c$ denota uma operação aritmética no domínio dos criptogramas. Entretanto, sistemas completamente homomórficos genéricos apresentam um desempenho impraticável para aplicações reais.

Em [Naehrig et al. 2011] os autores mostram que é possível evitar os problemas de desempenho dos sistemas completamente homomórficos, pois várias aplicações exigem apenas esquemas parcialmente homomórficos, suportando apenas alguns tipos de operações. Por apresentar desempenho muito superior, estes esquemas estão mais próximos de serem aplicáveis na prática. Os autores apresentam também uma implementação de um algoritmo cuja segurança se baseia no problema *Ring learning with errors (RLWE)* o qual suporta soma e multiplicação modulares com desempenho praticável. Paillier [Paillier 1999] e ElGamal exponencial [Gamal 1985] são exemplos de cifras parcialmente homomórficas eficientes e probabilísticas com nível de segurança IND-CPA.

## 4. Estendendo a Confidencialidade do DEPSPACE

Esta seção discute a proposta de incremento de segurança ao DEPSPACE através de uma arquitetura para compartilhamento de chaves e aplicação destes esquemas criptográficos ao protocolo. Como trata-se de um serviço voltado para sistemas distribuídos, o DEPSPACE pode ser modelado como grupos de processos clientes utilizando o espaço de tuplas, este por sua vez pode estar mantido em servidores não necessariamente confiáveis. Nossa primeira proposta consiste em retirar dos servidores qualquer informação sobre as chaves, pois como visto na Seção 2.1.2, um conluio de servidores maliciosos poderia recuperar as chaves por completo e comprometer toda a segurança do sistema.

Para evitar tal comprometimento, propomos que as chaves sejam de conhecimento apenas dos clientes, os quais devem previamente compartilhá-las utilizando um algoritmo de criptografia de chave pública com um mecanismo que disponha de proteção contra o ataque *Man-in-the-middle*, como sugerido em [Khader and Lai 2015]. Com isso podem co-existir grupos independentes de processos, cada um compartilhando seu próprio conjunto de chaves de acordo com a necessidade de compartilhamento de informações.

### 4.1. Protocolo

A análise apresentada na Seção 2.1.2 mostra que, devido à forma de como foi projetado e construído, o DEPSPACE está sujeito a uma série de ataques simples. Para contornar este problema, propomos a redução (ou eliminação) do uso de campos classificados como *públicos* ou *comparáveis* e a adoção da seguinte classificação:

- **Comparável Determinístico (CD)**, os quais serão cifrados utilizando um algoritmo determinístico de criptografia simétrica (Seção 3.1);
- **Ordenável (OR)**, os quais serão cifrados utilizando um algoritmo simétrico de criptografia com preservação da relação de ordem, como o OPE (Seção 3.2);
- **Operável (OP)**, os quais serão cifrados utilizando um algoritmo homomórfico ou parcialmente homomórfico (Seção 3.3).

Assim, para gerar o *fingerprint* $t_h = \langle h_1, ..., h_m \rangle$ de uma tupla $t = \langle f_1, ..., f_m \rangle$ e seu vetor de proteção $v_t = \langle v_1, ..., v_m \rangle$, a função $fingerprint(t, v_t) = t_h$ calcula cada $h_i = E_i(K_i, f_i)$, utilizando a cifra $\mathcal{E}_i$ e a chave compartilhada $k_i$ de acordo com o tipo $v_i$. Caso $f_i = *$, então $h_i = *$, ou ainda se $v_i = PR$ então $h_i = PR$. O cliente então armazena no espaço de tuplas o *fingerprint* $t_h$ e a tupla cifrada $t' = E_s(K_s, t)$ utilizando uma chave $K_s$ e uma cifra simétrica $\mathcal{E}_s$. Com isso, continua garantido que a tupla $t$ e um molde $\bar{t}$ combinam se $t_h$ combina com $\bar{t}_h$. Então ao encontrar um *fingerprint* $t_h$ que combina com um *fingerprint* $\bar{t}_h$ de um molde $\bar{t}$ utilizado para consulta, o servidor retorna a tupla cifrada $t'$ que será decifrada pelo cliente utilizando a função $D_s(K_s, t') = t$ com a chave compartilhada $K_s$.

Devido aos motivos citados na Seção 2.1, aconselhamos fortemente a preferência pelo uso da classificação *comparável determinístico* proposta ao invés da *comparável*, pois utilizando uma cifra simétrica determinística ao invés do *hash*, um atacante precisaria de acesso à chave secreta para cifrar um texto, impossibilitando o ataque de pré-imagem. Entretanto, esta classificação deve ser utilizada com cuidado (Seção 2.1.2), pois como esta cifra revela a igualdade de textos claros, em um domínio muito pequeno utilizando poucas informações externas, o conteúdo destes campos pode ser facilmente descoberto. Por exemplo, ao utilizá-lo para cifrar um campo que contém o sexo em uma base que conhecidamente tem mais homens que mulheres, um atacante pode ver que existem apenas dois textos cifrados possíveis e concluir que aquele com mais ocorrências se refere ao sexo masculino. Portanto, este campo deve ser utilizado somente para campos que sirvam como índices, com grande quantidade de possíveis valores e que não sejam em si dados sensíveis, como ID's, endereços de *e-mail*, nomes de nós de processos, dentre outros.

Alguns algoritmos criptográficos utilizam modos de operação com Vetores de inicialização (IV) randomizados como forma de prover criptografia probabilística, tendo como opção de fixar estes IV's (em zero, por exemplo) como forma de prover criptografia determinística. Nestes casos, recomenda-se que, ao invés do uso do IV fixo, seja utilizado uma função pseudo-aleatória (PRF) sobre a mensagem utilizando uma chave $k_1$, produzindo uma saída pseudo-aleatória $r = F(k_1, m)$ e então utilizar $r$ como IV da função de encriptação utilizando uma chave $k_2$, produzindo $c = E(k_2, m; r)$. Como a PRF gera a mesma saída para a mesma mensagem, o algoritmo continua determinístico, entretanto para mensagens diferentes a PRF gera saídas diferentes e portanto IV's diferentes para cada mensagem, alcançando o nível de segurança IND-DCPA [Boneh and Shoup 2015].

A classificação *ordenável*, por sua vez, possibilita uma gama de funcionalidades sobre as tuplas, como ordená-las por um campo deste tipo, realizar consultas de intervalo, selecionar máximos e mínimos, dentre outras. Da forma com que o DEPSPACE foi proposto, para se possibilitar este tipo de consulta, o campo precisaria estar classificado como *público*, ou caso os dados em questão sejam sensíveis, perde-se totalmente as funcionalidades classificando-se o campo como *privado*. Este tipo de funcionalidade é extremamente útil na execução de consultas mais complexas nos servidores [Distler et al. 2015]. Portanto, tanto o nível de segurança quanto o poder das buscas são aumentados. Entretanto há que se tomar bastante cuidado na escolha dos campos que utilizarão esta cifra, pois como demonstrado por [Naveed et al. 2015] um campo cifrado desta forma torna-se extremamente vulnerável aos ataques de inferência se todos os valores possíveis de um domínio estão presentes na coleção de dados. Por exemplo, se o campo referir-se à idade de pacientes em um hospital, que sabidamente tenha pacientes de todas as idades de 1 a 100 anos, os dados podem ser totalmente revelados devido a esta associação.

Como mostrado na Seção 3.2, o estado-da-arte deste tipo de algoritmo é o apresentado em [Lewi and Wu 2016]. Porém, por necessitar que parte dos dados sejam salvos no cliente para fazer as operações de comparação o algoritmo não se mostrou interessante para a aplicação no espaço de tuplas, pois neste caso os "clientes" são processos executando em um ambiente distribuído. Portanto, sugerimos que seja utilizado o algoritmo de OPE apresentado por [Boldyreva et al. 2012] e que a saída deste seja aplicada ao algoritmo prático de ORE da forma como sugerido por [Chenette et al. 2015]. Esta composição não causa grande impacto no desempenho e como provado pelos autores, faz com que o sistema atinja o nível de segurança IND-OCPA, vazando apenas o primeiro *bit*

que diferencia os valores comparados, neste caso da forma cifrada em OPE do número original ao invés do número propriamente dito.

Por último, a classificação de um campo como *operável* permite a computação sobre dados cifrados. Para isso, sugere-se o uso de uma cifra homomórfica ou parcialmente homomórfica (Seção 3.3), de acordo com a necessidade da aplicação. Um campo como este traria substancial incremento de funcionalidade ao DEPSPACE como a possibilidade de atualizar valores utilizados para sincronia de processos temporalmente desacoplados, sem revelar aos servidores o estado em que cada um se encontra. Para campos como este, que geralmente necessitam de apenas um tipo de operação (como soma/subtração), não se faz necessário o uso de uma cifra completamente homomórfica, uma cifra parcialmente homomórfica seria suficiente e não comprometeria de sobremaneira seu desempenho. Cabe ressaltar que mesmo utilizando algoritmos mais eficientes, um campo cifrado desta forma será o ponto crítico do desempenho do sistema, portanto propõe-se utilizá-lo somente quando extremamente necessário. Além disso, para estes campos, o cliente deve considerar o valor atualizado contido no *fingerprint* e não aquele da tupla.

Um dos fatos mais importantes a se considerar é que todas estas funcionalidades oferecidas por estas cifras podem ser utilizadas em uma busca ou operação sem que o servidor onde as tuplas estão armazenadas precise conhecer total ou parcialmente as chaves de decifração para os dados em questão, possibilitando que uma busca segura seja realizada mesmo se as tuplas estiverem armazenadas em um servidor não confiável.

**Análise de Segurança.** Como citado na Seção 2.1.2, uma cifra vulnerável a um ataque mais fraco será considerada como tendo um nível inferior de segurança, mesmo que seja resistente a um ataque mais forte. Tomando isso em conta, apresentamos na Tabela 1 o nível de segurança intrínseco a cifra utilizada em cada campo, ordenados do menor para o maior nível. Os níveis IND-OCPA e IND-DCPA são ambos flexibilizações do nível IND-CPA, porém além de igualdade, a cifra ordenável revela uma relação (ordem) entre textos cifrados diferentes, sendo portanto classificada em nível inferior ao IND-DCPA.

| PU | CO | OR | CD | OP | PR |
|---|---|---|---|---|---|
| Inseguro | Pré-imagem/colisão | IND-OCPA | IND-DCPA | IND-CPA | IND-CCA2 |

**Tabela 1. Nível de segurança dos campos no DEPSPACE**

A segurança do sistema depende, portanto, do adequado uso de cada cifra, levando em consideração o tipo e o quão sensível é o dado em cada campo. Por exemplo, deve-se priorizar o uso destas cifras funcionais para campos utilizados para indexação e referência, evitando o uso em campos que contém dados mais críticos ou sensíveis. Mantendo todos os campos cifrados evita-se principalmente os ataques de inferência que de uma maneira computacionalmente mais simples causam grandes estragos mesmo a sistemas extremamente complexos.

## 5. Trabalhos Relacionados

Dentre os vários sistemas desenvolvidos com o intuito de introduzir segurança e/ou tolerância a falhas em espaços de tuplas, o DEPSPACE [Bessani et al. 2008] e a sua versão estendida para coordenação distribuída [Distler et al. 2015] são os únicos que consideram tanto mecanismos para tolerância a falhas (replicação) quanto para segurança (criptografia e mecanismos de controle de acesso). Alguns sistemas agregam apenas mecanismos

de replicação [Bakken and Schlichting 1995, Xu and Liskov 1989], enquanto outros utilizam apenas controle de acesso [Busi et al. 2003, De Nicola et al. 1998, Vitek et al. 2003]. Outros sistemas merecem destaque por introduzir suporte a tolerância a falhas através do conceito de transações [T. J. Lehman et al. 2001, GigaSpaces 2016, JavaSpaces 2016]. Estes trabalhos sobre segurança para espaço de tuplas têm um foco muito limitado, pois consideram apenas ataques simples (acesso inválido ou usam mecanismos criptográficos que não são fortes o suficiente para garantir a segurança da informação armazenada).

Mecanismos criptográficos mais robustos foram usados no contexto de banco de dados visando prover confidencialidade e proteção contra vazamento de informações através do processamento de consultas em bases de dados cifradas [Popa et al. 2011, Alves and Aranha 2016]. Nestes trabalhos, os autores apresentam estratégias de atribuição de esquemas diferentes de criptografia de acordo com a especificidade dos dados armazenados em cada campo e a categoria das consultas esperadas para tal campo, como igualdade, comparação, continência de palavras em textos, dentre outros. Em campos cifrados de forma homomórfica, por exemplo, podem ser efetuadas operações de UPDATE sem decifrar os dados. Porém, em campos de somente consulta, onde comparações utilizando inequações ($\leq$ ou $\geq$) são realizadas, cifras do tipo OPE e ORE são empregadas, apresentando um bom desempenho sem grande perda no nível de segurança. As soluções propostas nestes trabalhos possibilitam o emprego de mecanismos criptográficos mais robustos também no contexto de espaço de tuplas, visando mitigar problemas de segurança relacionados com as aplicações que utilizam este paradigma de programação.

## 6. Conclusão e Trabalhos Futuros

Este trabalho discutiu alguns problemas encontrados na implementação de segurança, principalmente privacidade, em dados compartilhados através de espaços de tuplas, os quais estão principalmente relacionados com o acesso às tuplas através do conteúdo dos seus campos. Para contorná-los, propõe-se a integração deste paradigma de programação com esquemas atuais e robustos de criptografia.

Como trabalhos futuros, pretende-se implementar tanto as extensões sugeridas ao DEPSPACE quanto uma aplicação que faça uso de toda sua potencialidade, possibilitando analisar o desempenho destes protocolos, embora os aspectos de segurança sejam mais relevantes. Outro trabalho futuro é o projeto e desenvolvimento de um protocolo para buscas eficientes em dados cifrados, de forma que o desempenho do sistema seja melhorado sem que as propriedades de segurança sejam afetadas.

## Referências

Alves, P. G. M. R. and Aranha, D. F. (2016). A framework for searching encrypted databases. In *XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSEG 2016)*, pages 142–155.

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Bakken, D. E. and Schlichting, R. D. (1995). Supporting Fault-Tolerant Parallel Programing in Linda. *IEEE Transactions on Parallel and Distributed Systems*, 6(3):287–302.

Bellare, M., Desai, A., Pointcheval, D., and Rogaway, P. (1998). Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology - CRYPTO*

*'98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 26–45.

Bellare, M., Kohno, T., and Namprempre, C. (2004). Breaking and provably repairing the ssh authenticated encryption scheme: A case study of the encode-then-encrypt-and-mac paradigm. *ACM Trans. Inf. Syst. Secur.*, 7(2):206–241.

Bessani, A. N., Alchieri, E. P., Correia, M., and da Silva Fraga, J. (2008). DEPSPACE: A byzantine fault-tolerant coordination service. *European Conference on Computer Systems - EuroSys*, pages 163–176.

Bessani, A. N., Correia, M., Fraga, J. S., and Lung, L. C. (2006). Sharing memory between Byzantine processes using policy-enforced tuple spaces. In *Proceedings of 26th IEEE International Conference on Distributed Computing Systems - ICDCS 2006*.

Boldyreva, A., Chenette, N., Lee, Y., and O'Neill, A. (2012). Order-preserving symmetric encryption. Cryptology ePrint Archive, Report 2012/624. `http://eprint.iacr.org/2012/624`.

Boneh, D., Lewi, K., Raykova, M., Sahai, A., Zhandry, M., and Zimmerman, J. (2014). Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/834. `http://eprint.iacr.org/2014/834`.

Boneh, D. and Shoup, V. (2015). A graduate course in applied cryptography. `https://crypto.stanford.edu/~dabo/cryptobook/draft_0_2.pdf`. Acessado: 2016-11-30.

Busi, N., Gorrieri, R., Lucchi, R., and Zavattaro, G. (2003). SecSpaces: a Data-Driven Coordination Model for Environments Open to Untrusted Agents. In *Electronic Notes in Theoretical Computer Science*, volume 68.

Castro, M. and Liskov, B. (2002). Practical Byzantine fault-tolerance and proactive recovery. *ACM Transactions Computer Systems*, 20(4):398–461.

Chenette, N., Lewi, K., Weis, S. A., and Wu, D. J. (2015). Practical order-revealing encryption with limited leakage. Cryptology ePrint Archive, Report 2015/1125. `http://eprint.iacr.org/2015/1125`.

De Nicola, R., Ferrari, G. L., and Pugliese, R. (1998). KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330.

Distler, T., Bahn, C., Bessani, A., Fischer, F., and Junqueira, F. (2015). Extensible distributed coordination. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15.

Gamal, T. E. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472.

Gelernter, D. (1985). Generative Communication in Linda. *ACM Transactions on Programing Languages and Systems*, 7(1):80–112.

GigaSpaces (2016). GigaSpaces Homepage. Disponível em `http://www.gigaspaces.com/`.

JavaSpaces (2016). JavaSpaces guide. Disponível em `http://www.oracle.com/technetwork/articles/java/javaspaces-140665.html`.

Khader, A. S. and Lai, D. (2015). Preventing man-in-the-middle attack in diffie-hellman key exchange protocol. In *22nd International Conference on Telecommunications, ICT 2015, Sydney, Australia, April 27-29, 2015*, pages 204–208.

Lewi, K. and Wu, D. J. (2016). Order-revealing encryption: New constructions, applications, and lower bounds. In *ACM Conference on Computer and Communications Security (ACM CCS)*.

Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. (1996). *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

Morais, E. and Dahab, R. (2012). Encriptação homomórfica. In *Minicursos do XII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSEG 2012)*, pages 151–195.

Naehrig, M., Lauter, K., and Vaikuntanathan, V. (2011). Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 113–124, New York, NY, USA. ACM.

Naveed, M., Kamara, S., and Wright, C. V. (2015). Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 644–655.

Paillier, P. (1999). 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, pages 223–238, Berlin, Heidelberg. Springer-Verlag.

Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. (2011). CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100.

Schneider, F. B. (1990). Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.

Schoenmakers, B. (1999). 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*, pages 148–164.

T. J. Lehman et al. (2001). Hitting the distributed computing sweet spot with TSpaces. *Computer Networks*, 35(4):457–472.

Veríssimo, P. (2016). Dialogue on cyber policies between brazil and the eu: prospecting threats and opportunities of the cyberspace. Dialogue on Cyber Policies.

Vitek, J., Bryce, C., and Oriol, M. (2003). Coordination processes with Secure Spaces. *Science of Computer Programming*, 46(1-2):163–193.

Xu, A. and Liskov, B. (1989). A design for a fault-tolerant, distributed implementation of Linda. In *Proc. of the 19th Symposium on Fault-Tolerant Computing*, pages 199–206.