Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Provendo Privacidade no Modelo de Coordenação por Espaço de Tuplas

Edson Floriano de Sousa Junior

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientador
Prof. Dr. Eduardo Adilo Pelinson Alchieri

Coorientador
Prof. Dr. Diego Freitas Aranha

Brasília
2018

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Provendo Privacidade no Modelo de Coordenação por Espaço de Tuplas

Edson Floriano de Sousa Junior

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Prof. Dr. Eduardo Adilo Pelinson Alchieri (Orientador)
CIC/UnB

Prof. Dr. Alysson Neves Bessani
LaSIGE/University of Lisbon, Lisbon, Portugal

Prof. Dr.a Priscila Solís
CIC/UnB

Prof. Dr. Bruno Luiggi Macchiavello Espinoza
Coordenador do Programa de Pós-graduação em Informática

Brasília, 16 de fevereiro de 2018

Dedicatória

Dedico este trabalho aos meus pais, Edson e Edelves, que mesmo em face às dificuldades enfrentadas pela família, não mediram esforços para proporcionar aos seus filhos uma educação de qualidade, sempre nos cobrando o nosso melhor.

Agradecimentos

Agradeço a Deus, por tudo o que era, o que é e que há de ser em minha vida.

À minha esposa, Déborah, por estar ao meu lado mesmo nos momentos mais difíceis, me incentivando e me dando todo o suporte necessário, não apenas para este trabalho, mas para minha vida.

Às famílias, minha e de minha esposa, pela compreensão nos momentos em que me privei de estar com eles para me dedicar aos estudos.

Ao meu orientador, Prof. Dr. Eduardo Alchieri, pela oportunidade oferecida e pelas palavras certas em cada momento durante o projeto.

Ao meu co-orientador, Prof. Dr. Diego Aranha, por ter me inspirado e instigado a ir além e dar os meus primeiros passos na busca pelo conhecimento científico.

Aos superiores e colegas de trabalho do presente e de outrora pelo apoio e suporte prestados para tornar possível a realização deste curso. "AD ASTRA ET ULTRA"

Aos amigos e amigas que direta ou indiretamente contribuíram com a realização deste feito.

Resumo

A coordenação entre processos se configura como um grande desafio no desenvolvimento de sistemas distribuídos. Um dos modelos utilizados para realização de coordenação entre processos temporal e espacialmente desacoplados é por Espaços de Tuplas, que consiste em uma implementação de memória compartilhada que provê armazenamento e recuperação de objetos de dados chamados tuplas. Buscas de tuplas são realizadas de modo associativo, através do conteúdo de seus campos. Este tipo de acesso pode impedir que haja privacidade dos dados armazenados, tornando-as vulneráveis a uma série de ataques, já que os servidores precisam acessar dados em claro para realizar buscas.

Com o objetivo de sanar este problema, este trabalho apresenta propostas visando prover privacidade no sistema DepSpace, um sistema de coordenação que implementa mecanismos de tolerância a falhas e confiabilidade combinadas com aspectos de segurança. A ideia principal é utilizar esquemas criptográficos de computação privativa, que possibilitam a busca e computação sobre dados cifrados. Assim, os servidores podem operar sobre dados sem tomarem conhecimento dos mesmos. O sistema resultante além de prover privacidade, aumenta suas funcionalidades, tornando-se mais flexível. Apresentamos ainda uma análise de segurança do sistema com as melhorias propostas, juntamente com sua análise de desempenho, explicitando o impacto causado pelos algoritmos criptográficos.

Experimentos foram realizados aplicando as propostas à coordenação distribuída extensível, um modelo que utiliza computação dos dados nos servidores para tornar a coordenação mais ágil. Os resultados mostram uma redução de até 90% na latência do sistema e um aumento de até quase 9x na vazão (*throughput*) no processamento de mecanismos de coordenação em comparação à abordagem tradicional não extensível.

Palavras-chave: Espaço de Tuplas, segurança da informação, coordenação em sistemas distribuídos, computação privativa

Abstract

The coordination of distributed processes is a big challenge in the development of distributed applications. Tuple spaces provide a model for processes coordination that is decoupled in space and time. 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. This kind of access could impair user and data privacy, making these systems susceptible to several types of attacks since servers need to access plaintext data to search for tuples.

In order to deal with this problem, this work proposes mechanisms to provide privacy on DEPSPACE, a fault-tolerant coordination system that combines dependability and security properties. The main idea is to use privacy-preserving cryptography schemes, that allow search and computation over encrypted data. Consequently, servers could operate over data without knowing them. Beyond to provide privacy, the resulting system increases its functionalities, being more flexible. This work also presents a security analysis of the system with the proposed improvements, together with its performance analysis that shows the impact caused by the cryptographic algorithms.

A set of experiments was implemented applying this proposals for extensible distributed coordination, a model that uses data computing on the servers side to make the coordination faster. The results show that it could bring a reduction of up to 90% on the system latency and increase almost 9x its throughput on processing coordination mechanisms, comparing to the non-extensible traditional approach.

Keywords: Tuple space, information security, distributed systems coordination, privacy-preserving computing

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CDE Coordenação Distribuída Extensível.

DSM Distributed Shared Memory.

IV Initialization Vector.

OPE Order Preserving Encryption.

OPES Order Preserving Encryption Scheme.

ORE Order Revealing Encryption.

OTP One-time pad.

PRF Pseudo Random Function.

PRG Pseudo Random Generator.

PVSS Publicly Verifiable Secret Sharing.

SD Sistemas Distribuídos.

Capítulo 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 [5]. Intuitivamente, privacidade é entendida na perspectiva de uma entidade como a confidencialidade de suas informações sensíveis (dados e metadados) [72]. Esta entidade pode ser uma pessoa, uma organização, uma nação, etc. Como podemos observar, a privacidade está diretamente relacionada com a confidencialidade das informações.

1.1 Motivação

Atualmente existem vários fatores que aumentam o risco à segurança das aplicações [72]: *(i)* o mundo está se tornando uma infraestrutura imensa, interconectada e interdependente; *(ii)* existem muitos dados correlacionados entre si publicamente disponíveis; *(iii)* as entidades estão se expondo cada vez mais; e *(iv)* o número de vulnerabilidades de *software* está aumentando. Exemplos como o vazamento do site *Yahoo* em que dados como nomes, endereços de e-mail, números de telefone e datas de nascimento de pelo menos 500 milhões de usuários foram roubadas [38], são a prova de que mesmo grandes sistemas são passivos de sofrerem estes ataques.

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, ataques de inferência estatística [58] muitas vezes conseguem obter informações mantidas sob sigilo através da análise e correlação de dados de acesso público. Consequentemente, 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 (*Tuple Space - TS*) [13, 39], onde o acesso é feito

de maneira associativa, através do conteúdo dos campos das tuplas. Como exemplos de sistemas distribuídos que podem se beneficiar de um espaço de tuplas seguro, pode-se mencionar aplicações de alto nível como pregão eletrônico seguro [2] e aplicações que precisam de memória compartilhada distribuída [2, 14] ou estruturas de sincronização como contadores compartilhados e listas distribuídas [30].

Embora existam algumas propostas que visam adicionar propriedades de segurança neste modelo [13, 22, 28, 30, 73], a necessidade de acesso através do conteúdo dos campos faz com que as mesmas sejam suscetíveis a ataques capazes de quebrar a privacidade dos dados e de seus usuários. O principal problema a ser resolvido para prover privacidade neste modelo é que os servidores precisam ser capazes de selecionar tuplas com base em seu conteúdo, porém sem conhecê-lo.

Dentre as propostas existentes, o DEPSPACE [13] 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 (1) público – os dados são armazenados em claro e todas as partes ou processos podem acessá-los; (2) comparável – apenas um resumo (*hash*) do conteúdo do campo fica disponível; ou (3) privado – nenhuma informação é disponibilizada. Dada esta classificação, pelo menos um dos campos de cada tupla precisa ser público ou comparável para que o acesso a uma tupla seja possível, isto é, para que um servidor possa comparar tuplas e moldes (Section 3.3.1). Esta abordagem, apesar de trazer algum nível de confidencialidade, traz consigo um grande desafio ao desenvolvimento de aplicações: se muitos campos públicos e/ou comparáveis são usados, o sistema se torna vulnerável a ataques de correlação, de pré-imagem e de colisão; por outro lado, como os servidores não podem executar buscas e comparações em dados privados, as possibilidades de buscas por tuplas podem ser significativamente reduzidas para se preservar a segurança da informação, limitando seu uso no desenvolvimento de aplicações distribuídas.

Visando contornar estes problemas, este trabalho 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 adequados a este cenário, as extensões propostas preservam as propriedades de segurança do sistema e proporcionam maior flexibilidade nas buscas ao modelo tradicional de coordenação por espaço de tuplas.

Finalmente, é proposta a aplicação do sistema resultante no modelo de coordenação distribuída extensível [30] conferindo-lhe propriedades de segurança e privacidade. Neste cenário, dados são processados nos servidores, evitando as perdas relacionadas às falhas causadas por processamento concorrente entre os clientes e à transmissão de dados entre clientes e servidores. Experimentos mostram que, apesar do impacto no desempenho causado pelos algoritmos criptográficos, o processamento de contadores compartilhados e

filas distribuídas utilizando as propostas resultam em ganhos de desempenho significativos em relação às suas abordagens tradicionais.

1.2 Objetivos

O objetivo geral deste trabalho é projetar e desenvolver protocolos para prover segurança, principalmente privacidade, em dados compartilhados através de espaço de tuplas.

Dentro do objetivo geral, os seguintes objetivos específicos são observados:

- Estudar os conceitos relacionados com Sistemas Distribuídos (SD) focando em coordenação de processos;
- Estudar os conceitos de Segurança da Informação e os diversos protocolos de criptografia, buscando algoritmos apropriados ao cenário de comunicação por tuplas;
- Analisar a segurança do sistema de coordenação DEPSPACE [13] e propor extensões ao sistema visando torná-lo mais seguro;
- Implementar as soluções propostas e analisar o desempenho e nível de segurança do sistema resultante;
- Aplicar as soluções propostas no desenvolvimento de protocolos de coordenação distribuída extensível, que façam uso de toda sua potencialidade.

1.3 Metodologia

Este trabalho tem natureza empírica baseado em uma pesquisa aplicada de objetivos exploratórios. Utilizou-se as abordagens qualitativa em relação aos métodos de segurança que se deseja implementar, e quantitativa em relação às métricas de desempenho resultantes da aplicação destes métodos. Quanto às técnicas e procedimentos, realizou-se uma pesquisa de caráter bibliográfico e experimental.

A pesquisa bibliográfica foi realizada visando o entendimento tanto do "estado-da-arte" dos algoritmos criptográficos a serem utilizados quanto das características intrínsecas aos sistemas distribuídos orientados a coordenação. Todo o estudo foi guiado pela interação interdisciplinar da aplicação dos paradigmas de segurança da informação ao campo de pesquisas em sistemas distribuídos. O estudo sistemático de questões ligadas à segurança de informação proporcionou a adoção de um olhar crítico utilizado na análise das características do sistema em questão.

A pesquisa experimental foi então realizada, testando algoritmos criptográficos na busca pelos que melhor se encaixam ao ambiente de coordenação por espaço de tuplas.

Foram utilizadas bibliotecas já bem estabelecidas para cada algoritmo realizando as devidas adaptações para integrá-las o DEPSPACE. Traçou-se então uma análise qualitativa dos níveis de segurança alcançados com a proposta e quantitativa a cerca do impacto no desempenho causado pelos algoritmos criptográficos.

Para validação dos resultados, foram implementados dois mecanismos largamente utilizados para coordenação de processos. Os resultados de desempenho dos mecanismos foram então medidos empiricamente e comparados quantitativamente com o modelo tradicional de coordenação com vistas de se estabelecer o nível de melhora de desempenho da aplicação do modelo extensível de coordenação.

1.4 Organização do texto

Visando atingir os objetivos citados, este trabalho apresenta no Capítulo 2 uma visão geral sobre aspectos de segurança, bem como algumas técnicas utilizadas para atingir seus objetivos. Já no Capítulo 3 são expostos os conceitos básicos de Sistemas Distribuídos e tolerância a falhas, evoluindo até os sistemas de coordenação por Espaço de Tuplas e como os aspectos de segurança se encaixam neste cenário. O Capítulo 4 contém então a proposta desenvolvida durante o trabalho, juntamente com sua análise de segurança e desempenho. O Capítulo 5 comenta o modelo de Coordenação Distribuída Extensível e apresenta a aplicação das propostas do capítulo anterior aos contatores compartilhados e filas distribuídas, trazendo sua análise de desempenho neste cenário. Finalmente, o Capítulo 6 traz as conclusões do trabalho e as visões de trabalhos futuros.

Capítulo 2

Segurança da Informação e Criptografia

Não é de hoje a noção de que a informação é um dos ativos mais importantes de qualquer organização. Desde os primórdios, civilizações como os egípcios protegiam seus segredos de seus inimigos. Entretanto, no mundo tão digital e conectado que vivemos hoje, o dito popular "o segredo é a alma do negócio" ganha cada vez mais importância também no âmbito computacional, visto que incidentes como o já citado do portal *Yahoo* [38] acabam causando, além de uma enorme perda de credibilidade, prejuízos financeiros e morais tanto às instituições alvo dos ataques quanto aos usuários que tiveram seus dados expostos. Neste capítulo será visto o que significa segurança em um sistema de computação, quais os principais tipos de ataque aos quais estão sujeitos e algumas ferramentas e noções de segurança utilizados para prevenir problemas decorrentes destes ataques.

2.1 Segurança da Informação

Segundo Menezes et al. [55], para prover **Segurança da Informação**, sistemas devem atingir alguns objetivos em relação a estas informações. Dentre outros citados pelo autor, destacam-se a **Confidencialidade** e a **Integridade** que são apresentados na Tabela 2.1, juntamente com a **Disponibilidade**, apresentada por Avizienis et al. [5], que definem ainda que um sistema é dito **seguro** se fornece estas três propriedades. Resumindo, em um sistema seguro, temos a informação certa, disponível na hora certa para a pessoa certa.

Além da academia, no mundo dos negócios a segurança da informação é tão importante que foi padronizada internacionalmente pela diretiva ISO/IEC 17799:2005 [45] (posteriormente corrigida e substituída pelas diretivas ISO/IEC 27002:2005 [46] e 27002:2013 [47]),

a qual trata da implementação dos controles e procedimentos que visam proteger a informação das diversas ameaças minimizando as vulnerabilidades a que podem ser exploradas.

Confidencialidade	Manutenção dos dados em segredo, exceto para aqueles que estejam autorizados a ter conhecimento. Ex.: Mensagens enviadas de A para B não podem ser legíveis a E.
Integridade	Garantia de que os dados não estejam sendo alterados por pessoas não autorizadas. Ex.: B deve estar apto a identificar se a mensagem enviada por A foi modificada por E.
Disponibilidade	Prontidão para prover o serviço correto. Ex.: A consegue acessar o serviço provido por B sempre que precisa fazê-lo.

Tabela 2.1: Propriedades de Segurança da Informação. [5, 55]

2.2 Criptografia

Para se alcançar as propostas citadas, várias técnicas são utilizadas. **Criptografia** é um termo que se refere a um conjunto de técnicas que buscam prover principalmente confidencialidade, mas que também está relacionada a integridade e aspectos como autenticação de entidade e origem dos dados em questão. Formalmente, define-se **criptografia** como a área de pesquisa que desenvolve técnicas matemáticas que possibilitam comunicações seguras na presença de adversários maliciosos. [55]. Sejam $\mathcal{M}$, $\mathcal{C}$ e $\mathcal{K}$ os conjuntos finitos de possíveis *textos claros*, possíveis *textos cifrados* (ou *criptogramas*) e possíveis *chaves* (ou *espaço de chaves*), respectivamente, um **criptossistema** (ou uma *cifra*) $\mathcal{E}$ definido sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ será composto pelas funções de *encriptação* (ou *cifração*) $Enc(k_e, m) : (\mathcal{K} \times \mathcal{M}) \rightarrow \mathcal{C}$ e *decifração* $Dec(k_d, c) : (\mathcal{K} \times \mathcal{C}) \rightarrow \mathcal{M}$ tais que para cada $m \in \mathcal{M}$ e $k_e \in \mathcal{K}$, temos que existe uma $k_d \in \mathcal{K}$ tal que $Dec(k_d, Enc(k_e, m)) = m$. Uma cifra será dita *probabilística* se para entradas fixas de chave $k \in \mathcal{K}$ e mensagem $m \in \mathcal{M}$ da função de cifração, a saída $c = Enc(k, m)$ pode assumir valores diferentes. Caso contrário, a cifra será dita *determinística* [20].

Segundo Hankerson et al. [27], os criptossistemas podem ser divididos em dois grandes tipos: os de **criptografia simétrica** e os de **criptografia assimétrica**. No primeiro caso, as chaves k_d e k_e podem ser eficientemente calculadas uma a partir da outra, sendo seu caso mais extremo quando $k_e = k_d$. São exemplos deste tipo de criptografia os sistemas *Data Encryption Standard* (DES) [33] e o *Advanced Encryption Standard* (AES) [32]. Ainda com criptografia simétrica, estas entidades podem usar um Código de Autenticação de Mensagem (MAC) tal como HMAC [9] para garantir a integridade e autenticação de mensagem.

Porém, este tipo de cifra, apesar de sua alta eficiência, esbarra nos chamados problemas de *distribuição de chaves* e de *gerenciamento de chaves* [27], os quais consistem em:

1. Necessidade de um canal ao mesmo tempo confidencial e autenticado para a distribuição de chaves, sendo necessário, em grande parte das vezes a utilização de um canal físico considerado seguro, como um portador confiável (pessoa credenciada, por exemplo) para fazê-lo.

2. Em uma rede com N entidades, cada uma precisa manter uma chave diferente com cada uma das $N - 1$ entidades da rede. Além do mais, como uma chave é compartilhada entre duas (ou mais) entidades, estes algoritmos não podem ser usados para prover esquemas elegantes de *assinatura digital*, visto que é impraticável fazer distinção entre as ações de diferentes titulares da mesma chave secreta.

Estes problemas podem ser tratados com a utilização do segundo tipo de criptossistemas, os de criptografia assimétrica, também conhecidos como de **criptografia de chave pública** [29]. Neste caso, segundo Washington [74], as partes comunicantes, suponhamos, Alice e Bob, não precisam de um contato prévio em canal seguro para troca de chaves. Bob disponibiliza uma *chave pública* que será utilizada por Alice para cifrar mensagens destinadas a Bob. Este, por sua vez, possui uma *chave privada* (mantida em segredo) que o habilita a decifrar os textos cifrados com a sua chave pública. Neste tipo de algoritmo, deve ser computacionalmente impraticável deduzir o texto em claro ou a chave privada a partir da chave pública, visto que esta última é conhecida por todos. O mais famoso sistema de chave pública conhecido é o RSA [61], que se baseia na dificuldade da fatoração de inteiros grandes.

Independente do tipo de criptografia utilizada, a tarefa de mensurar o nível de segurança de um sistema não é trivial. Como se verá a seguir, não é viável atingir **segurança incondicional** contra todo tipo de ataque *matematicamente possível*, portanto o nível de segurança de um sistema deverá ser estabelecido em termos do esforço computacional necessário para quebrá-lo (**segurança computacional**) ou na dificuldade de se resolver o problema matemático no qual o sistema se baseia (**segurança demonstrável**) [69].

Para afirmar que um sistema é *computacionalmente seguro*, deve ser provado que o melhor algoritmo para quebrar este sistema necessita de um número de passos que excede o poder computacional do adversário para ser executado. Enquanto que para se afirmar que um sistema é *demonstradamente seguro*, mostra-se que se um sistema pode ser quebrado de alguma forma, então o problema matemático conhecidamente difícil no qual ele se baseia poderia também ser resolvido de modo eficiente. Neste caso, o que existe é uma redução a um problema subjacente e não uma prova absoluta de segurança.

2.2.1 Segredo Perfeito

Seja $\mathcal{E}$ uma cifra definida sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ e sejam $\mathbf{K}$, $\mathbf{X}$ e $\mathbf{Y}$ variáveis aleatórias nos espaços de chaves, textos claros e criptogramas, respectivamente, com $\mathbf{K}$ e $\mathbf{X}$ independentes. Definimos $\mathbf{Pr}[\mathbf{K} = k]$ como a probabilidade de se selecionar uma chave k para cada $k \in \mathcal{K}$. Da mesma forma, para cada $x \in \mathcal{M}$ e $y \in \mathcal{C}$ temos que $\mathbf{Pr}[\mathbf{X} = x]$ é a probabilidade prévia de se selecionar um texto claro x e $\mathbf{Pr}[\mathbf{Y} = y]$ a probabilidade posterior de ter como resultado um criptograma y.

Definimos ainda $\mathbf{Pr}[x|y]$ como a *probabilidade condicional* de que $\mathbf{X}$ assuma o valor x, dado que $\mathbf{Y}$ assume o valor y. Temos que $\mathbf{X}$ e $\mathbf{Y}$ são independentes se e somente se $\forall x \in \mathcal{M}, \forall y \in \mathcal{C}, \mathbf{Pr}[x|y] = \mathbf{Pr}[x]$, ou seja, dado um criptograma y a probabilidade de que o texto claro correspondente seja x é idêntica à probabilidade prévia de se selecionar x no conjunto $\mathcal{M}$. Um sistema criptográfico que atende a esta premissa é dito ter **Segredo Perfeito**. Isso significa que o criptograma não deve fornecer qualquer ideia do texto em claro correspondente a ele. Assim, o **Teorema de Shannon** estabelece que para fornecer segredo perfeito, um criptossistema com $|\mathcal{K}| = |\mathcal{M}| = |\mathcal{C}|$ deve poder utilizar qualquer chave $k \in \mathcal{K}$ com a mesma probabilidade $1/|\mathcal{K}|$ e ainda que para todo $m \in \mathcal{M}$ e $c \in \mathcal{C}$ exista uma única chave tal que $c = Enc(k, m)$ [69].

O problema com esta definição é que o conjunto de chaves deve ser tão grande quanto o conjunto de textos possíveis e cada chave destas tão grande quanto o texto claro que se objetiva cifrar, algo totalmente inviável. Um exemplo disso é a cifra conhecida como *One-time pad (OTP)* que consiste em efetuar a operação *XOR* entre o texto claro m e a chave k selecionada aleatoriamente no espaço de chaves ($c = m \oplus k$). O texto cifrado resultante não oferece qualquer indicação sobre o texto claro correspondente, entretanto se uma mesma chave for utilizada mais que uma vez, ela se torna vulnerável. Isto porque de posse de um par (m, c), um atacante pode obter a chave apenas calculando $k = m \oplus c$, utilizando então esta chave para decifrar outras mensagens [20].

Portanto, é necessário estabelecer outros critérios para analisar a segurança de um criptossistema e estabelecer níveis de segurança que sejam possíveis de serem implementados.

2.2.2 Segurança Semântica

Continuando a busca por uma definição de segurança que torne os sistemas viáveis, é necessário um enfraquecimento da definição de segurança baseada no poder computacional do adversário e no quanto ele pode ter acesso e ainda assim o sistema permanecer seguro.

Com esta finalidade, define-se então a **segurança semântica** de um sistema conforme segue. Para todo **adversário eficiente** $\mathcal{A}$, isto é, polinomial, uma cifra $\mathcal{E}$ definida

sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ e uma instância desta cifra, chamada de **oráculo**, utilizando uma chave k aleatoriamente selecionada no conjunto uniformemente distribuído $\mathcal{K}$, definimos o **experimento** $\boldsymbol{b}$ ou $Exp(b)$, com $b \in \{0, 1\}$ como:

- $\mathcal{A}$ computa duas mensagens $m_0, m_1 \in \mathcal{M}$, de tamanhos iguais e envia para o oráculo;
- O oráculo escolhe ao acaso um $b \in \{0, 1\}$, seleciona uma chave k, computa o criptograma $c_b = Enc(k, m_b)$ e o envia para $\mathcal{A}$;
- Com base apenas neste criptograma, $\mathcal{A}$ tenta deduzir a qual experimento esta resposta se refere, ou seja, se $b = 0$ ou $b = 1$, gerando como saída b', conforme ilustrado na Figura 2.1.

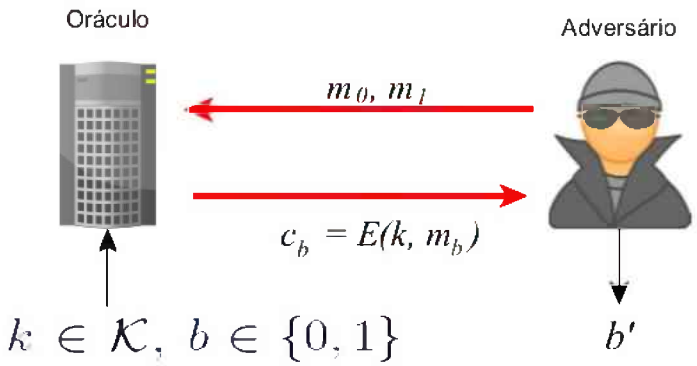


Figura 2.1: $Exp(b)$ e a segurança semântica.

Com base nisso, seja W_b o evento em que a $\mathcal{A}$ gera $b' = 1$ como saída do $Exp(b)$, definimos a **vantagem** (*advantage*) de $\mathcal{A}$ em relação a $\mathcal{E}$ como:

$$Adv[\mathcal{A}, \mathcal{E}] := |\mathbf{Pr}[W_0] - \mathbf{Pr}[W_1]|.$$

Uma cifra $\mathcal{E}$ será dita **semanticamente segura** se para todo adversário eficiente $\mathcal{A}$ o valor $Adv[\mathcal{A}, \mathcal{E}]$ for desprezível [20]. Em outras palavras, se a probabilidade de que $\mathcal{A}$ gere 1 como saída for aproximadamente a mesma para qualquer dos experimentos 0 e 1, então $\mathcal{A}$ não tem condições de distinguir $Exp(0)$ de $Exp(1)$. Essa definição traz consigo uma forte noção de confidencialidade, e seu conceito será utilizado para apresentar as diversos modelos de indistinguibilidade baseados em classes de ataque.

2.2.3 Classes de Ataque e Indistinguibilidade

Segundo Menezes et al. [55], os ataques aos esquemas criptográficos buscam obter o texto claro ou a chave de decifração através dos seguintes métodos:

- *Ciphertext-only attack (COA)* ou Ataque de texto cifrado conhecido: 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. Este ataque é muito semelhante ao anterior, exceto pelo fato de que aqui o atacante pode escolher quais textos claros serão cifrados.
- *Adaptative chosen-plaintext attack (CPA2)* ou Ataque adaptativo de texto claro escolhido: 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)* ou Ataque adaptativo de texto cifrado escolhido: 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. Voltemos ao exemplo da cifra OTP. Como dito, nesta cifra o criptograma não fornece qualquer informação sobre o texto claro, logo o primeiro ataque relacionado acima não é possível. Entretanto, o OTP é vulnerável ao ataque de texto claro conhecido, visto que de posse de um par texto claro (m) e criptograma (c), um atacante pode efetuar a operação $m \oplus c$ e obter a chave k utilizada.

Este é o motivo pelo qual no OTP uma mesma chave nunca deve ser utilizada para cifrar mais que uma mensagem, pois uma vez descoberta a chave utilizada para cifrar várias mensagens, um atacante poderia decifrar todos os demais criptogramas facilmente. Outra possibilidade seria, de posse de dois criptogramas c_1 e c_2 cifrados com a mesma

chave k, um atacante pode efetuar a operação $c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$ e utilizar análise de frequência do idioma para inferir as mensagens originais m_1 e m_2, o chamado **ataque *two-time pad*** [20].

Outros conceitos importantes que devem ser apresentados são os de **pré-imagem** e **colisão**. Dada uma função de resumo criptográfico (*hash*) $h : X \to Y$, uma função será dita **resistente à pré-imagem** se, dado um resumo $y \in Y$ fixo, é computacionalmente inviável encontrar um $x \in X$ tal que $h(x) = y$. Se, para um resumo y e uma mensagem x tais que $y = h(x)$, for computacionalmente inviável encontrar uma mensagem $x' \neq x$ tal que $h(x') = h(x) = y$, então diz-se que a função é **resistente à segunda pré-imagem**. Já uma colisão ocorre quando existem duas mensagens quaisquer $x, x' \in X$ tais que $h(x) = h(x')$. Uma função para a qual é computacionalmente inviável encontrar tais mensagens é dita **resistente a colisões** [69]. Os mesmos conceitos podem ser facilmente estendidos para qualquer função criptográfica $f : (\mathcal{M} \times \mathcal{K}) \to \mathcal{C}$, apenas com a diferença que nestes casos é necessário o uso de uma chave para produzir o criptograma.

Para estabelecer os níveis de segurança contra os demais ataques utilizaremos o conceito de indistinguibilidade, o qual formaliza a incapacidade de um adversário em obter alguma informação a respeito do texto claro a partir do texto cifrado [10]. De acordo com o mesmo autor, para todo adversário eficiente $\mathcal{A}$, uma cifra $\mathcal{E} = (Enc, Dec)$ 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 $\mathcal{A}$ e submetidas a um oráculo que responde com o **criptograma** $c_i = Enc(k, m_{ib}) \in \mathcal{C}$, chamado "desafio", para alguma chave $k \in \mathcal{K}$ selecionada aleatoriamente e $b \in \{0, 1\}$, a probabilidade de que $\mathcal{A}$ possa distinguir se $c_i = Enc(k, m_{i0})$ ou $c_i = Enc(k, m_{i1})$ com uma chance maior que 50% 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 de decifração que dado um texto cifrado $c_j \notin \{c_1, ..., c_{j-1}\}$ responde com o **texto claro** $m_j = Dec(k, c_j)$ correspondente e da mesma forma a probabilidade de que $\mathcal{A}$ possa distinguir se $c_i = Enc(k, m_{i0})$ ou $c_i = Enc(k, m_{i1})$ com uma chance maior que 50% 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.

Voltando ao conceito de segurança semântica, dizemos que uma cifra é semanticamente segura contra o ataque ATK $\in$ {CPA, CCA, CCA2} se o valor $Adv[\mathcal{A}, \mathcal{E}] = |\mathbf{Pr}[W_0] - \mathbf{Pr}[W_1]|$ para o ataque ATK é desprezível. Verificamos portanto que os conceitos de indistinguibilidade e segurança semântica são equivalentes e portanto se um sistema oferece indistinguibilidade contra um determinado ataque, pode ser dito *seguro* (semanticamente) contra este mesmo ataque ou *ATK-seguro*.

2.2.4 Segurança de Cifras Determinísticas

Consideremos agora uma cifra determinística $\mathcal{E}$. Cifras deste tipo possuem a característica de vazar igualdade de textos claros quando cifrados sob a mesma chave, ou seja, $m_0 = m_1 \iff Enc(k, m_0) = Enc(k, m_1)$. Essa característica pode ser utilizada para realizar buscas por igualdade em dados cifrados, como mostrado em [3, 60]. Pode-se verificar que uma cifra como esta não pode ser CPA-segura. Isso porque um atacante pode enviar para o oráculo de decifração duas cópias da mesma mensagem m_0 na primeira rodada, recebendo a forma cifrada de m_0, $c_0 = Enc(k, m_0)$, independente de se tratar do experimento $b = 0$ ou 1 e em seguida enviar juntamente com m_0 uma mensagem qualquer m_1. Ao receber a resposta do oráculo, basta comparar c_b e c_0 para responder com 100% de certeza se $b = 0$ ou $b = 1$, ou seja a vantagem do atacante $\mathcal{A}$ sobre a cifra $\mathcal{E}$, $Adv[\mathcal{A}, \mathcal{E}] = |\mathbf{Pr}[W_0] - \mathbf{Pr}[W_1]| = |0 - 1| = 1$. Portanto, faz-se necessário o estabelecimento de uma definição de segurança específica para este tipo de cifra, uma flexibilização da definição de IND-CPA para o caso em que a cifra é determinística, o que foi proposto mais recentemente por Bellare [11]:

- **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 $\mathbf{A}$, 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 $c_i = Enc(k, m_{ib}) \in \mathcal{C}$ para alguma chave k selecionada aleatoriamente em $\mathcal{K}$ e $b \in \{0, 1\}$, a probabilidade de que $\mathbf{A}$ possa distinguir se $c_i = Enc(k, m_{i0})$ ou $c_i = Enc(k, m_{i1})$ com uma chance maior que 50% é desprezível.

Ou seja, por esta definição, uma mensagem m cifrada deterministicamente com uma chave k estará segura enquanto nenhum atacante tiver conhecimento do par $(m, Enc(k, m))$. Já para as cifras probabilísticas (ou aleatorizadas), conhecer um par $(m, Enc(k, m))$ não compromete a segurança da mensagem m cifrada posteriormente com a mesma chave k, pois diversos são os valores que $c' = Enc(k, m)$ pode assumir.

2.2.5 Ataques de Inferência

Apesar dos ataques citados na Seção 2.2.3 serem os principais considerados pela literatura, eles não se limitam a esta lista. Uma infinidade de outros ataques pode ser possível dependendo das características do sistema. Por exemplo, Naveed et al. [58] 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, como sexo, raça e risco de mortalidade, foram descobertos em 60% dos hospitais, enquanto que mais de 80% dos dados cifrados com uma cifra que preserva a ordem dos dados (Seção 2.3.1), como idade e nível de severidade de doença, foram recuperados em 95% dos hospitais.

Segundo os autores, estes ataques foram possíveis devido ao mal uso de técnicas criptográficas. Isso porque uma cifra determinística, devido aos motivos citados na Seção 2.2.4, jamais deve ser utilizada para cifrar campos que tenham um espaço tão pequeno de valores possíveis, principalmente se todos eles ocorrem na coleção de dados. Por exemplo, em hospital que conhecidamente tenha mais pacientes do sexo feminino que masculino, basta contabilizar os dois valores possíveis para o campo sexo (ou procurar o campo que tem apenas dois valores possíveis, caso o nome do campo também esteja cifrado) e verificar qual valor mais ocorre e com isso concluir que se trata do "feminino". Da mesma forma, campos cifrados com cifras com preservação de ordem, que além de determinísticas vazam a ordem, podem ter todo seu conteúdo exposto. Como exemplo, ainda no hospital hipotético, caso seja público o conhecimento de que todas as idades de 0 a 100 ocorrem no campo idade o próprio algoritmo entrega ao atacante os valores ordenados e portanto, basta uma correlação rápida para obter todos os valores em claro correspondentes.

Ataques de inferência cada vez mais complexos são facilitados hoje em dia, na medida em que cada vez mais dados são disponibilizados publicamente pelas pessoas em geral nas diversas mídias e redes sociais. Por este motivo, torna-se interessante manter cifrado todo dado possível, mesmo que não seja sigiloso ou sensível, de modo a evitar que este dado acabe servindo como ligação entre os dados sigilosos e os publicamente disponíveis.

2.3 Computação Privativa

Após implementar um bom nível de segurança através das técnicas e algoritmos mencionados até aqui, um sistema pode se tornar inviável se toda vez que precisar realizar uma busca necessitar decifrar toda uma base, ou se precisar decifrar mesmo que apenas uma entrada cada vez que necessitar alterá-la. Em vistas de evitar todo esse retraba-

lho, o impacto em desempenho que ele implica e ainda preservar a segurança, alguns trabalhos propõem o uso da **Computação Privativa**, que em suas diversas abordagens pode possibilitar desde a execução de funções como a comparação e consequentemente a ordenação de dados cifrados [17, 18, 19, 24] até a execução de operações como soma e multiplicação sobre estes dados, sem a necessidade de decifrá-los ou conhecer as chaves de decifração [56, 57, 61].

Com este poder em mãos, torna-se factível tanto realizar buscas como fazer alterações sobre dados cifrados sem que os servidores nos quais estes estão armazenados tenham conhecimento destes dados, fato relevante tendo em vista o crescimento de modelos como a computação em nuvem, em que cada vez mais usuários armazenam informações em servidores dos quais não é possível garantir total idoneidade. Vejamos agora as características de algumas dessas abordagens.

2.3.1 Cifras de Ordem

Proposto por Agrawal et. al [1], o *Order Preserving Encryption Scheme (OPES)* se trata de um esquema criptográfico simétrico que preserva a relação de ordem para dados numéricos, ou seja, para todo i e j e para toda chave k, $Enc(k,i) < Enc(k,j) \iff i < j$. A ideia por trás deste esquema é proporcionar uma forma de se realizar comparações entre números cifrados o que consequentemente habilita consultas como intervalos e máximos e mínimos em bases de dados cifrados, bem como realizar operações como *COUNT*, *GROUP BY* e *ORDER BY* nestas consultas.

É evidente que 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 com tal propriedade, Boldyreva et al. [17] apresentam seu próprio esquema, chamando-o apenas de *Order Preserving Encryption (OPE)*. Este algoritmo retorna como saída um inteiro de precisão arbitrária (*BigInteger* em JAVA) determinístico que preserva exatamente a mesma ordem que os números em claro, com uma distância pseudo-aleatória entre dois números cifrados. Neste mesmo trabalho, os autores apresentam uma definição de segurança voltada para este tipo de cifra:

- **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 $\mathbf{A}$ 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 $c_i = Enc(k, m_{ib}) \in \mathcal{C}$ para alguma chave k selecionada aleatoriamente em

$\mathcal{K}$ e $b \in \{0,1\}$, a probabilidade de que **A** possa distinguir se $c_i = Enc(k, m_{i0})$ ou $c_i = Enc(k, m_{i1})$ é desprezível [17].

Os autores mostram ainda que nenhum algoritmo determinístico pode atingir este nível de segurança, pelos mesmos motivos mostrados na Seção 2.2.4. Seu algoritmo utiliza funções pseudo-aleatórias (PRF's) e geradores pseudo-aleatórios (PRG's), desfrutando de um nível flexibilizado 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 [17, 18].

Outro problema de construções deste tipo, é que para atender à premissa de ordenação dos textos cifrados, estes devem ser numéricos, reduzindo bastante o espaço de possíveis valores. Para contornar este problema, o tamanho do criptograma deve ser aumentado, porém com isso, o desempenho do sistema acaba sendo comprometido. Boneh et al. [19] então propõem o *Order Revealing Encryption (ORE)*, uma construção que não impõe restrições em relação aos criptogramas, ao invés disso, disponibiliza uma função capaz de comparar dois criptogramas. Portanto, neste tipo de algoritmo não é mais possível comparar criptogramas diretamente como nas cifras OPE. Os autores provam ainda que este esquema atinge o melhor nível de segurança semântica possível, compatível com o IND-OCPA.

Entretanto, devido ao paradigma no qual se baseia, o algoritmo proposto por Boneh et. al se mostrou inviável em termos de desempenho. Posteriormente, Chenette et al. [24] propõem um algoritmo de ORE cujo desempenho possibilita que seja praticável, onde se tem a exata noção de que dados são vazados pela cifra, neste caso, o primeiro *bit* que diferencia os valores comparados. Mais recentemente, Lewi e Wu [52] apresentam um algoritmo de ORE praticável e resistente aos ataques de inferência.

O algoritmo ORE de Lewi e Wu se destaca por não ser determinístico, adicionando mais segurança ao esquema. Como mostrado por Alves e Aranha, este algoritmo funciona muito bem em aplicações de Banco de Dados cifrados [3, 53].

2.3.2 Criptografia Homomórfica

A Criptografia Homomórfica [56, 57, 61] tem sido pesquisada como uma forma de se realizar operações sobre dados cifrados sem a necessidade de decifrá-los ou conhecer a chave de decifração obtendo os mesmos resultados que se teria caso a operação correspondente tivesse sido executada com os dados em claro. Isso se deve ao fato de que, em uma cifra homomórfica, dadas duas mensagens quaisquer m_1 e m_2, uma função de cifração Enc e uma

chave k, existem operações $\circ_m$ e $\circ_c$ tais que $Enc(k, m_1) \circ_c Enc(k, m_2) = Enc(k, m_1 \circ_m m_2)$, 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 [57]. Naehrig et al. [57] mostram que é possível evitar os problemas de desempenho dos sistemas completamente homomórficos, pois várias aplicações exigem suporte de apenas alguns tipos de operações, o que pode ser desempenhado por esquemas parcialmente (*somewhat*) homomórficos de criptografia. Por apresentar desempenho muito superior, estes esquemas estão mais próximos de serem aplicáveis na prática. Os autores apresentam também uma implementação de um algoritmo que suporta soma e multiplicação modulares com desempenho praticável.

Como exemplos de cifras parcialmente homomórficas eficientes e probabilísticas com nível de segurança IND-CPA, podemos citar:

- O algoritmo de Paillier [59], cuja segurança é baseada no problema da fatoração de inteiros e tem seu homomorfismo aditivo definido como $Enc(k, m_1) \cdot Enc(k, m_2) = Enc(k, m_1 + m_2)$. Ou seja, ao multiplicar dois criptogramas, o resultado é a forma cifrada da soma dos dois textos claros correspondentes; e
- ElGamal [31], cuja segurança é baseada na dificuldade do problema do Logaritmo Discreto. Este algoritmo em sua proposta original tem seu homomorfismo multiplicativo definido como $Enc(k, m_1) \cdot Enc(k, m_2) = Enc(k, m_1 \cdot m_2)$, mas pode ser adaptado para sua versão Exponencial, a qual possui homomorfismo aditivo definido como $Enc(k, m_1) \cdot Enc(k, m_2) = Enc(k, m_1 + m_2)$. Por simplicidade, omite-se aqui as questões relacionadas às chaves efêmeras utilizadas no algoritmo, mais informações podem ser encontradas em [31].

2.3.3 Busca em Bancos de Dados Cifrados

Popa et al. [60] apresentam em seu trabalho a abordagem que serviu como base para um grande número de trabalhos que buscam prover confidencialidade no processamento de consultas em dados cifrados [3, 53, 54]. A abordagem, ambientada em banco de dados relacionais, consiste em atribuir esquemas diferentes de criptografia de acordo com os tipos dos dados armazenados e as consultas esperadas para cada campo.

Os autores propõem o uso de cifra determinística para consultas de igualdade, cifra de ordem para consultas de comparação e intervalo e cifra homomórfica para valores numéricos que podem ser atualizados (apenas soma). Soma-se a estas o uso de uma cifra simétrica padrão para dados mais sensíveis, sobre os quais não se pode realizar qualquer

tipo de computação. Combinações de primitivas criptográficas podem ser usadas, como em camadas, para fornecer níveis de segurança e funcionalidades diferentes.

Macedo et al. [54] utilizam o mesmo princípio, porém no contexto de bancos de dados *NoSQL*, provendo um *framework* completo para preservação de privacidade neste tipo de banco. Exceto pela cifra homomórfica, os autores aplicaram basicamente os mesmos algoritmos, impossibilitando, portanto, atualizações de dados cifrados diretamente no servidor.

Alves e Aranha [3, 53], por sua vez, apesar de utilizarem o mesmo princípio baseado em cifras de ordem e homomórficas, conseguem atingir um nível de segurança mais alto por não utilizarem algoritmos determinísticos em seu *framework*. A cifra de ordem OPE, por exemplo, foi substituída pela ORE [52]. Apesar de utilizarem as operações de álgebra relacional para descrever seu *framework*, os autores afirmam que o conceito pode ser estendido para outras classes de bancos de dados mantendo a mesma estrutura.

2.4 Considerações

Os sistemas utilizam diversos meios para prover as propriedades relacionadas com a segurança da informação, dentre os quais destaca-se a criptografia. Mesmo na inviabilidade de fornecer segurança incondicional, esta ciência busca fornecer um nível suficiente de segurança tomando o devido cuidado de não tornar o sistema impraticável em termos de desempenho em face à grande variedade de ataques possíveis. Com o advento da computação privativa este desafio torna-se ainda mais instigante, pois esta acrescenta soluções para diminuir a necessidade de se decifrar e cifrar dados quando se deseja fazer consultas e/ou operações. Com isso diminuem-se as vulnerabilidades que possibilitam ataques de inferência, os quais de uma maneira simples podem causar grandes transtornos para empresas e clientes.

Um olhar sobre o estado da arte em busca sobre dados cifrados, nos mostra que capacidade de buscas e confidencialidade podem conviver sem que uma anule a outra, entretanto em um eterno compromisso entre desempenho e segurança.

Capítulo 3

Sistemas Distribuídos Baseados em Coordenação

A necessidade de usuários geograficamente espalhados em obter compartilhamento de recursos de forma eficiente e escalável, com garantias de disponibilidade e a baixo custo é uma das principais motivações para o desenvolvimento de **Sistemas Distribuídos (SD)**. Este capítulo faz uma breve exposição sobre sistemas deste tipo, apresentando os conceitos de tolerância a falhas e memória compartilhada e dentro destes, abordam-se o modelo de coordenação através de espaços de tuplas e algumas de suas principais implementações.

3.1 Introdução

Diversas são as definições de SD na literatura, cada uma com um enfoque um pouco diferente. Segundo Coulouris [25], um sistema distribuído é *"aquele no qual os componentes de hardware ou software localizados em computadores interligados em rede, se comunicam e coordenam suas ações apenas enviando mensagens entre si"* (p. 2). Esta definição apesar de genérica, retrata bem o que acontece no *backgroud* da implementação. Já Tanenbaum [71] expõe melhor o que é percebido pelo usuário, definindo também de maneira genérica que *"um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente"* (p. 1).

Em ambos os casos, percebemos que a execução paralela de atividades e o uso concorrente de recursos estão intimamente ligados aos sistemas distribuídos. Lamport [49], por sua vez, estabelece que *"um sistema distribuído consiste em uma coleção de* ***processos*** *distintos espacialmente separadas cuja comunicação é feita por troca de mensagens"*. Esta definição é mais abrangente por se referir aos processos como componentes essenciais de um sistema distribuído.

Entretanto, existem fatores complicadores, como a inevitável ocorrência de *falhas* dos componentes que integram o sistema e a *inexistência de relógio global*, isto é, devido aos limites de precisão, não é possível estabelecer uma noção exata de tempo comum a todos os computadores. Então para estabelecer a coordenação das ações, buscando garantir a *consistência* dos dados e a *coerência* da operações, a sincronização de tempo é toda feita através do envio de mensagens [49].

Tanto a abstração de sistema único para o usuário quanto a comunicação entre os processos são fornecidos por uma camada de software chamada ***middleware***. Esta camada é tão importante que pode ser considerada como o próprio sistema distribuído. O *middleware* é responsável por esconder do usuário diversos aspectos da implementação, como a heterogeneidade de hardware e sistemas operacionais dos computadores integrantes do SD. Da mesma forma, é ele quem orquestra a execução de processos entre os computadores, bem como sua comunicação e sincronização, garantindo a consistência citada no parágrafo anterior [71].

3.2 Tolerância a Falhas

Por ser composto por vários computadores independentes, um SD está suscetível a falhas que podem ocorrer de forma independente em cada uma das partes que o compõem. Cristian [26] apresenta uma classificação das diversas falhas que podem atingir os sistemas, como falhas por **omissão** (em que o servidor não consegue responder a uma requisição que lhe é enviada), por ***crash*** (em que após a primeira omissão o servidor passa a omitir todas as respostas até ser reiniciado), de **temporização** (em que a resposta do servidor se encontra fora do intervalo de tempo esperado) e de **resposta** (em que a resposta do servidor está incorreta).

Além destas, Shneider [63] apresenta as falhas por **parada** (em que em resposta a uma falha, o componente altera o seu estado de modo que os outros componentes possam detectar que uma falha ocorreu e então para) e Lamport et al. [50] apresenta as chamadas falhas **arbitrárias** ou **Bizantinas** (em que o servidor pode produzir respostas arbitrárias em momentos arbitrários). As falhas bizantinas são consideradas as mais graves, pois neste tipo de falha as respostas incorretas produzidas pelos servidores podem não ser identificadas como tais, podendo inclusive terem sido produzidas de forma maliciosa. Estão inclusas neste tipo de falha situações em que um ou mais servidores se organizam maliciosamente para produzir este tipo de resposta ou obter dados indevidamente.

Um sistema é dito **tolerante a falhas** quando, mesmo na presença de falhas, continua funcionando corretamente [71]. Para ser tolerante, um sistema deve ser capaz de detectar a falha e/ou de certa forma mascará-la de modo que o sistema como um todo possa

ainda prover seus serviços normalmente. Ocasionalmente, um sistema pode ser tolerante a falhas simultâneas de até k servidores, neste caso, será dito **k-tolerante a falhas**. Para prover tal tolerância, os principais mecanismos utilizados são a **redundância**, através da qual pode-se recuperar dados e operações perdidos ou incoerentes à partir de dados, operações ou equipamentos extras (como bits de verificação, repetição de operações ou equipamentos/processos sobressalentes) e a **replicação** que consiste em manter cópias completas de processos ou dados como forma de *backup* para recuperação em caso de perda ou falha.

3.3 Coordenação Distribuída

A comunicação entre processos é uma necessidade intrínseca aos sistemas distribuídos. Exclusão mútua distribuída e eleição de líderes são exemplos de situações que explicitam a necessidade de coordenação das ações entre os nós do sistema, visando a manutenção da consistência dos dados e a tolerância a falhas [25]. Para este fim, o uso de uma comunicação direta entre remetente e destinatário torna o sistema pouco versátil, ao passo que uma falha ocorrida em qualquer dos lados comunicantes, pode inviabilizar esta comunicação, impondo às partes a dificuldade de tratar explicitamente a falha ocorrida.

Situações como esta motivaram o desenvolvimento de comunicação indireta entre entidades em diversos sistemas cliente/servidor, através de uma entidade intermediária. Esta comunicação indireta pode ser feita fornecendo aos processos acesso a um espaço comum para troca de dados. Este tipo de comunicação fornece às partes comunicantes propriedades importantes conhecidas como **desacoplamento referencial (ou espacial)** e **temporal**, ou seja, as partes comunicantes não necessitam conhecer a identidade uma da outra (e consequentemente sua localização) e podem ter tempos de vida independentes. A primeira propriedade implica que os participantes podem ser a qualquer momento atualizados, substituídos, duplicados ou migrados, enquanto a segunda os dispensa da necessidade de estabelecer conexão simultânea para haver comunicação [25, 49, 71].

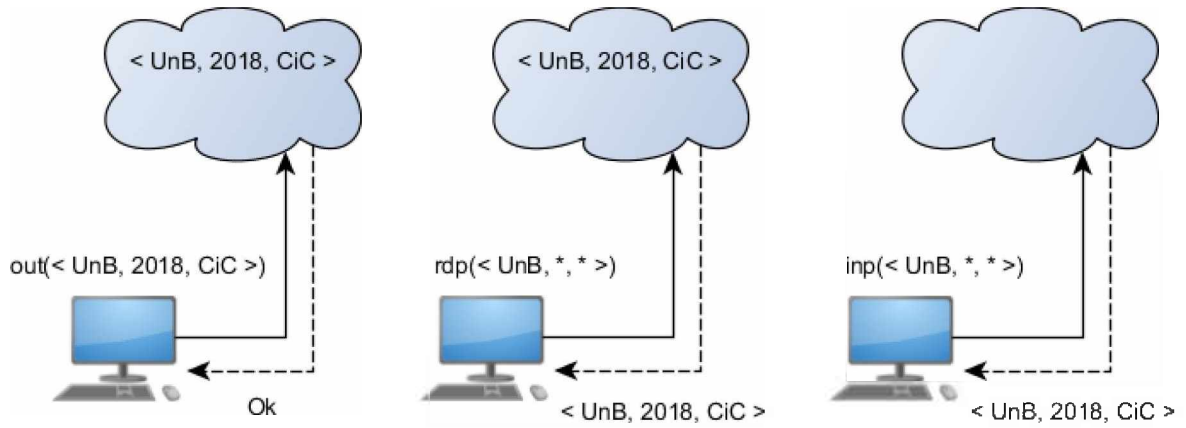
A necessidade de armazenamento temporário destes dados pode ser suprida utilizando uma abstração de **memória compartilhada**. Dentre as estratégias que fornecem esta abstração, destacam-se a **Memoria Compartilhada Distribuída** - *Distributed Shared Memory (DSM)* e o **Espaço de Tuplas**. Na primeira, os computadores acessam os dados na DSM diretamente através de seus endereços como se estivessem em sua memória física, sendo-lhes permitido manipulações como leitura, escrita e atualizações. O *middleware* é responsável por transformar a solicitação dos dados para a DSM de maneira transparente ao processo que a desencadeou [25]. A próxima seção se dedica a apresentar e discutir com maior profundidade a segunda estratégia, mais voltada para a coordenação.

3.3.1 Espaço de Tuplas

O espaço de tuplas, é uma estratégia de implementação de coordenação no qual os dados são dispostos em tuplas e armazenados na memória compartilhada, permitindo a coordenação de processos de um sistema distribuído desacoplada no espaço e no tempo [39]. Tuplas são sequências de um ou mais campos tipados, como <"joão", 1974>, <"MAT", "UnB">, <"CIC", "UnB", "Brasília">, <8, 7.5, "Não", "Masculino">.

Ao contrário do que ocorre no caso da DSM, as tuplas não são selecionadas diretamente por endereçamento mas sim indiretamente, através do casamento de padrões do próprio conteúdo da tupla. Para isso, são utilizados os chamados **moldes** (*templates*), que são tuplas que podem conter um ou mais campos indefinidos, isto é, campos coringa que podem ser casados (ou combinados) com qualquer valor e são simbolizados pelo caractere especial "*". Diz-se que uma tupla t e um *template* $\bar{t}$ combinam se e somente se ambos têm o mesmo número de campos e todos os valores e tipos dos campos definidos em $\bar{t}$ são iguais aos valores e tipos dos campos correspondentes em t. Então, por exemplo, o *template* <*, "UnB"> combina com a tupla <"MAT", "UnB"> mas não com a tupla <"CIC", "UnB", "Brasília">.

Neste modelo, as manipulações realizadas no espaço de tuplas consistem em invocações de três operações básicas: $out(t)$ que adiciona a entrada t no espaço de tuplas; $in(\bar{t})$, que realiza a leitura de uma tupla que combina com o *template* $\bar{t}$ e a remove do espaço de tuplas; $rd(\bar{t})$realiza a leitura de uma tupla que combina com o *template* $\bar{t}$, sem removê-la do espaço.



(a) $out(t)$: os servidores recebem uma tupla t submetida por um cliente, armazenam t no espaço de tuplas e retornam "ok".

(b) $rdp(\bar{t})$: servidores recebem um *template* $\bar{t}$, procuram uma tupla t que combine com $\bar{t}$ e a retornam mantendo-a no espaço de tuplas. Se nenhuma tupla for encontrada, retornam *null*.

(c) $inp(\bar{t})$: servidores recebem um *template* $\bar{t}$, procuram uma tupla t que combine com $\bar{t}$ e a retornam removendo-a do espaço do tuplas. Se nenhuma tupla for encontrada, retornam *null*.

Figura 3.1: Operações básicas do Espaço de Tuplas.

As operações *in* e *rd* são bloqueantes, isto é, se não houver uma tupla que combine com o *template* 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* que funcionam exatamente como as anteriores, a não ser pelo fato de retornarem mesmo não havendo uma tupla que combine com o *template* usado (indicando esta inexistência). A Figura 3.1 ilustra as operações de *out*, *rdp* e *inp*. Outras operações comumente adicionadas são [6, 13, 30, 66]: $cas(\bar{t}, t)$, que insere t no espaço se não tiver uma tupla t' que combine com o *template* $\bar{t}$ e retorna nulo, caso contrário retorna t'; $replace(t, t')$, que insere a tupla t' e remove (e retorna) a tupla t caso t esteja no espaço, caso contrário apenas retorna nulo; e $readAll(\bar{t})$, que retorna todas as tuplas que combinem com o *template* $\bar{t}$.

3.4 Trabalhos Relacionados

Apresentamos a seguir, alguns trabalhos que exploram os conceitos de segurança aplicados a espaço de tuplas.

3.4.1 Espaços de Tuplas com Propriedades de Segurança

Não é de hoje a preocupação com segurança em espaço de tuplas. Inicialmente, a preocupação era focada em garantir a disponibilidade de sistemas baseados em Linda [39]. Por isso, trabalhos como [6, 76] têm seu foco voltado a prover mecanismos de redundância e replicação para garantir tolerância a falhas a estes sistemas. Com a proposta de sistemas modelados para agentes móveis [28], surgiu a necessidade de focar em controle de acesso [15, 22, 73]. Nestes trabalhos, os agentes (processos) móveis visitam outros *sites* e coletam dados (tuplas) que precisam ser enviados ao seu proprietário. Como o agente está vulnerável a ataques enquanto transita, este não é confiável. Portanto, os agentes devem ser capazes de coletar os dados sem ter acesso ao seu conteúdo. Além do controle de acesso, primitivas criptográficas assimétricas são utilizadas em [15] para garantir que os agentes (que portam apenas chaves públicas) não possam decifrar e ter acesso aos dados coletados.

Paralelamente, a preocupação com a integridade dos dados mostrou-se na forma da introdução do suporte ao conceito de transações [37, 70], garantindo a atomicidade das operações. Nos dias atuais, sistemas robustos como GigaSpace [41] e JavaSpace [48] servem de base para a maior parte das aplicações baseadas em espaço de tuplas. Nestes trabalhos entretanto, a preocupação com a confidencialidade para espaço de tuplas têm um foco muito limitado, pois consideram apenas ataques simples (acesso inválido).

Dentre os sistemas de coordenação baseados em espaços de tuplas, o DEPSPACE [13] e a sua versão estendida para coordenação distribuída [30] se destacam por considerarem tanto mecanismos para tolerância a falhas (redundância e replicação) quanto mecanismos de criptografia e controle de acesso, visando com esta combinação atender as premissas básicas de segurança da informação (disponibilidade, integridade e confidencialidade) além de propriedades relativas à **confiabilidade**, isto é, as operações realizadas no espaço de tuplas fazem com que seu estado se modifique de acordo com a especificação.

Baseando-se no DEPSPACE, Silva e Correia [67] apresentam o *HomomorphicSpace*, uma extensão do DEPSPACE que utiliza alguns dos algoritmos criptográficos apresentados no Capítulo 2 para fornecer mecanismos de pesquisa e operações sobre dados cifrados no modelo de coordenação por espaço de tuplas. Para isso, os autores acrescentaram comandos ao sistema original que identificam consultas e operações deste tipo solicitadas pelos clientes. Paralelamente, neste trabalho seguimos a mesma linha com uma proposta que ao invés de implementar novos comandos, acrescenta novas classificações de campos ao sistema original. Apresentamos ainda uma completa análise de segurança do sistema pré e pós proposta e dos algoritmos implementados, além de levar em consideração os tempos de cifração e decifração de cada algoritmo na análise de desempenho.

3.4.2 DepSpace: Um sistema de Coordenação Tolerante a Falhas Bizantinas

Segurança é uma característica fundamental de sistemas confiáveis [5]. No contexto de um espaço de tuplas, os atributos de *confiabilidade*, *disponibilidade*, *integridade* e *confidencialidade* são necessários. O DEPSPACE [13] consiste na implementação de um espaço de tuplas que busca satisfazer estas propriedades por meio de diversas camadas, cada uma responsável pela concretização de uma funcionalidade ou propriedade distinta. Estas camadas são descritas a seguir (Figura 3.2), com uma maior ênfase para a camada responsável pela confidencialidade dos dados armazenados por ser extensivamente usada neste trabalho.

Pelo lado do cliente, a primeira camada, chamada de ***Proxy***, é responsável pela entrega da abstração ao cliente, transformando suas solicitações nas operações correspondentes no espaço de tuplas. A camada correspondente no lado do servidor é o próprio Espaço de Tuplas.

Logo após, a camada de **Controle de acesso** entra em ação para garantir que as operações serão executadas apenas por quem tem privilégio para tal. Para isso, o trabalho sugere utilizar, dentre outros esquemas, as Listas de Controle de Acesso — *Access Control Lists (ACL)* —, uma ligada ao TS, indicando quais processos têm permissão para inserir

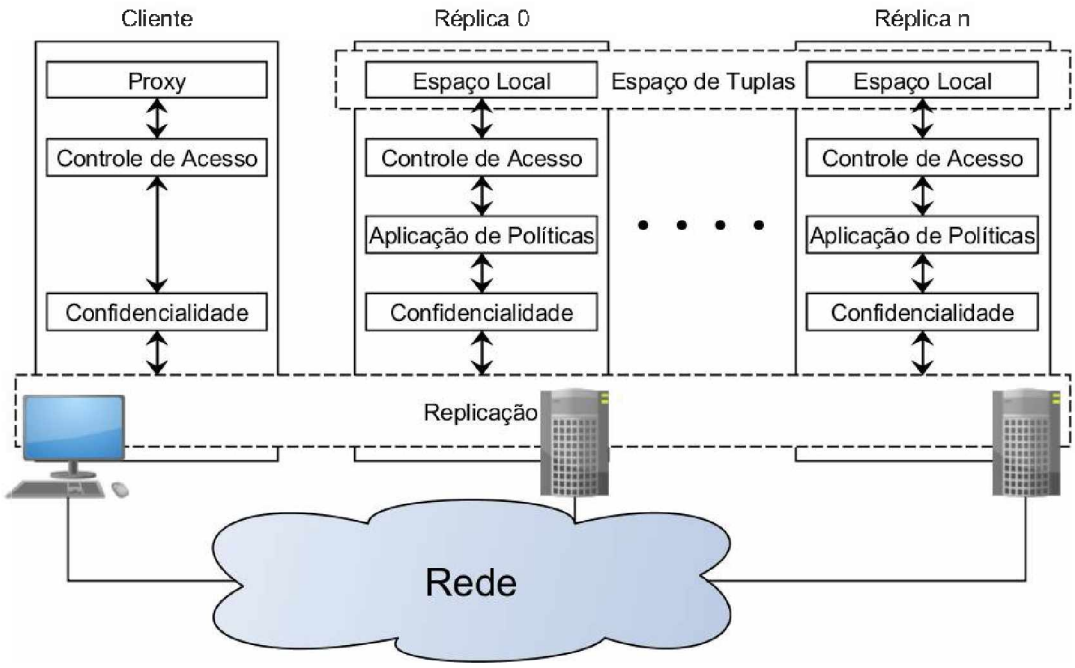


Figura 3.2: Camadas do DEPSPACE (adaptado de [13]).

tuplas, e outras duas ligadas a cada tupla, definindo quais processos têm permissão de leitura e de remoção sobre esta. Estas ACL's são aplicadas na camada correspondente no lado do servidor.

No cliente, ao receber uma solicitação por parte da camada de *proxy*, a camada de controle de acesso anexa as credenciais do usuário, para que possa ser verificado nas ACL's se este cliente possui privilégio para esta operação. Esta camada, juntamente com uma camada adicional de **Aplicação de Políticas** no lado do servidor, estão diretamente ligadas à manutenção da integridade e atuam como um primeiro passo na busca pela confidencialidade. Esta última camada possibilita a execução de uma política de acesso de granularidade fina [14] que leva em consideração três parâmetros: ((i)identificação do chamador; (ii) operação e argumentos; e (iii) tuplas atualmente armazenadas no espaço) para decidir se uma operação é aprovada ou negada. Estas políticas são definidas pelos usuários e carregadas nos servidores durante a inicialização do sistema.

Deixando por um instante a ordem entre as camadas, temos a camada de **Replicação**, responsável diretamente pela disponibilidade e integridade do sistema. Esta camada constitui a base do sistema por manter a consistência da réplica local do espaço de tuplas em cada um dos n servidores. Para suportar a até f servidores em falha, esta camada exige um número $n \geq 3f + 1$ réplicas. Assim, as réplicas corretas mascaram o comportamento das faltosas. Isso é feito através da **replicação máquina de estados (RME)** [64], a qual determina que todas as réplicas devem iniciar no mesmo estado e executar todas as

requisições na mesma ordem, produzindo a mesma saída. Para isso, é impreterível que as operações no espaço de tuplas sejam determinísticas, isto é, a mesma operação executada no mesmo estado inicial deve retornar o mesmo resultado em todas as replicas. Para alcançar este resultado é utilizado o BFT-SMaRt [12], uma biblioteca de RME tolerante a falhas bizantinas, para garantir que pelo menos $n - f \geq 2f + 1$ servidores retornem de forma idêntica.

Na camada de **Confidencialidade** o sistema busca atingir o atributo de mesmo nome além de contribuir indiretamente com os outros atributos de segurança. Para isso, são utilizadas técnicas de criptografia sobre as tuplas através do uso do esquema $(n, f+1)$ de compartilhamento de segredo publicamente verificável (*Publicly Verifiable Secret Sharing* – $(n, f+1)$-PVSS) [65] onde n e f são os números de servidores e falhas apresentados na camada de replicação. Através deste esquema, os clientes cifram as tuplas e geram um conjunto de n fragmentos (chamados *shares*) da tupla cifrada. O esquema PVSS é baseado em criptografia assimétrica e cada *share* é gerado utilizando a chave pública de um dos n servidores diferentes. Antes de serem enviados aos servidores, os *shares* são cifrados usando um algoritmo simétrico padrão, como *Triple-DES (3DES)* [33] e chaves compartilhadas entre o cliente e cada servidor. Para remontar a tupla, é necessário a combinação de $f + 1$ *shares*, o que torna impossível que um conluio de até f servidores maliciosos revele o conteúdo de uma tupla.

Como os servidores não podem acessar diretamente o conteúdo dos campos das tuplas (pois estão cifradas) esta camada emprega a combinação de uma classificação de cada campo com o uso de uma tupla especial chamada *fingerprint* t_h para cada tupla t. O *fingerprint* contém a mesma quantidade de campos que a tupla original, e será utilizado para realizar as buscas por casamento de padrões. Seus campos são preenchidos de acordo com uma classificação atribuída a cada campo da tupla original no ato de sua inserção no espaço, como segue:

- Público (PU): não utiliza criptografia e seu conteúdo original é usado no *fingerprint.*
- Comparável (CO): estes campos utilizam uma função de resumo criptográfico (*hash*).
- Privado (PR): Nenhuma informação referente ao conteúdo do campo da tupla original é inserida, apenas um caractere especial indicando sua classificação.

A coleção ordenada das classificações de cada campo de uma tupla t é chamada de *vetor de proteção* v_t. Por exemplo, a tupla $t =< 1, 3 >$ com vetor de proteção $v_t =< CO, PR >$ tem o seu primeiro campo classificado como comparável e o segundo como privado. Isso confere ao sistema um nível de confidencialidade dos dados críticos sem impedir a busca sobre estas tuplas.

Em síntese, uma operação de inserção (*out*) funciona da seguinte forma:

- O cliente c usa o esquema PVSS para gerar n *shares* da tupla t;
- c computa o *fingerprint* t_h de t usando o vetor de proteção v_t
- c cifra cada *share* com uma chave secreta compartilhada com cada servidor e um algoritmo de criptografia simétrica (um *share* por servidor);
- c usa o protocolo de replicação máquinas de estados para enviar uma requisição (inserção) aos servidores e aguarda por pelo menos $f + 1$ respostas para finalizar a execução da requisição. A requisição contém os *shares* cifrados, uma prova de que estes *shares* são válidos e o *fingerprint* da tupla;
- Cada servidor que recebe esta requisição extrai o *share* correspondente e verifica se este é válido utilizando a prova recebida;
- Cada servidor então armazena o *share* recebido e envia uma confirmação ao cliente como resposta.

As operações de leitura e remoção (*rdp* e *inp*) são feitas utilizando casamento de padrões entre os *fingerprints* das tuplas e dos *templates*, ou seja, uma tupla t e um *template* $\bar{t}$ combinam se cada campo definido no *fingerprint* de $\bar{t}$ combina com o correspondente no de t. O seguinte passo-a-passo demonstra como o protocolo funciona:

- O cliente c computa o *fingerprint* para o *template* de acordo com a classificação dos seus campos (o *fingerprint* de um campo indefinido é o próprio caractere coringa);
- c usa o protocolo de replicação máquina de estados para enviar uma operação de leitura/remoção para os servidores contendo o *fingerprint* gerado;
- Cada servidor que recebe esta requisição seleciona deterministicamente uma tupla tal que seu *fingerprint* combine com o *fingerprint* recebido;
- Se esta é uma operação de remoção (*inp*) esta tupla é removida do espaço;
- Cada servidor responde ao cliente com uma mensagem cifrada simetricamente que contém o *share*, o *fingerprint* e a prova de que o *share* é válido;
- c decifra a resposta e a verifica, aguardando por pelo menos $f+1$ respostas válidas, ou seja, com o mesmo t_h e cuja função de verificação dos *shares* tenha retornado verdadeiro;
- c combina $f + 1$ *shares* corretos recuperando a tupla t;
- c verifica se o *fingerprint* usado é válido para a tupla recuperada. Se for válido, a operação é finalizada. Caso contrário, um procedimento de reparação é executado para remover os dados inválidos do espaço e a operação é repetida.

3.4.3 Outras Abordagens para Coordenação Distribuída

Além de espaço de tuplas, outros modelos são utilizados por diversos sistemas para realizar as tarefas de coordenação. É o caso, por exemplo, do sistema *ZooKeeper* [44], que utiliza o modelo hierárquico de nós para prover um serviço de coordenação que visa a escalabilidade na leitura de dados. Estes nós, chamados *znodes*, são os objetos de dados manipulados pelos usuários através da API fornecida pelo sistema.

Outro sistema largamente utilizado para esta funcionalidade é o *Chubby* [21], que utiliza *locks* para implementar um sistema de coordenação cuja interface se assemelha à de um sistema de arquivos. O sistema, que serve de base para o *Google File System* [40], abre mão de uma alta performance para priorizar a disponibilidade e a confiabilidade na sincronização de atividades dos clientes.

Estes são apenas alguns exemplos de sistemas para coordenação distribuída. Esta é uma área vasta de pesquisa com diversas propostas utilizando várias abordagens.

3.5 Considerações

Neste capítulo falou-se sobre sistemas distribuídos, apresentando aspectos ligados à tolerância a falhas e coordenação distribuída. Diversos sistemas de coordenação por espaços de tuplas foram apresentados, onde foram abordados os aspectos de segurança fornecidos por cada um. Um enfoque foi dado no DepSpace, um sistema que visa fornecer os aspectos de confiabilidade, integridade, disponibilidade e confidencialidade tornando-se um sistema tolerante a falhas bizantinas com aspectos de segurança. Finalmente, são expostas rapidamente outras abordagens para coordenação distribuída.

Capítulo 4

Provendo Privacidade no Modelo de Coordenação por Espaço de Tuplas

Este capítulo apresenta a proposta de aplicação de esquemas criptográficos robustos ao sistema DEPSPACE, incrementando suas propriedades de segurança e privacidade ao mesmo tempo que aumenta sua flexibilidade nas buscas. Para isso, inicialmente apresenta-se uma análise de segurança do DEPSPACE que pautará as propostas apresentadas a seguir. Uma análise de segurança da proposta também é apresentada, juntamente com experimentos que analisam o desempenho das soluções propostas.

4.1 Análise de Segurança do DepSpace

Esta seção aponta algumas vulnerabilidades apresentadas pelo DEPSPACE com base na análise de sua definição. O sistema em questão apresenta pelo menos dois problemas notáveis, além de algumas questões que serão discutidas.

O primeiro problema, levantado pelos próprios autores, consiste no fato de que os campos comparáveis utilizam o algoritmo SHA-1, que apesar de permitir a seleção de tuplas sem a necessidade de decifrá-las, é vulnerável a ataques de colisão. Um exemplo de colisão contra a função SHA-1 foi mostrado possível em [68], onde foram gerados dois arquivos PDF diferentes com exatamente o mesmo *hash*. Além disso, se o domínio de valores possíveis é limitado e conhecido, estes campos serão vulneráveis ao ataque de pré-imagem. Isso porque um adversário sempre pode obter qualquer quantidade desejada de entradas e suas respectivas saídas simplesmente calculando seus *hashes*. Somado a isso, a existência de campos públicos pode proporcionar a um atacante a possibilidade de correlação entre os dados do sistema e uma base pública favorecendo os ataques de inferência (Seção 2.2.5).

O segundo problema diz respeito à funcionalidade das buscas no sistema. Por utilizar classificações muito limitadas, os dados sempre estão ou expostos/vulneráveis (Público/Comparável) ou totalmente sem funcionalidades para se manter seguros (Privado). Tal fato, como veremos no próximo capítulo, se torna um problema de segurança quando aplicado à coordenação extensível [30].

Outras questões que podem ser levantadas, dizem respeito ao esquema $(n,\ f+1)$-PVSS [65] utilizado para cifrar as tuplas. O esquema produz n fragmentos, chamados *shares* de modo que são necessários $f+1$ *shares* corretos para reconstituí-las. Estes *shares* são então distribuídos entre os servidores e graças a este esquema, o sistema suporta uma quantidade de até f de servidores em falha. Entretanto, para cifrar algo aleatório como uma tupla, o esquema requer o uso de um algoritmo simétrico que utiliza como chave um segredo gerado e distribuído em forma dos *shares*. Isso desencadeia dois problemas: Primeiro, os fragmentos das chaves estão sob o poder dos servidores. Em um cenário de possíveis falhas bizantinas, uma quantidade de servidores maior que f podem entrar maliciosamente em um acordo para conseguirem obter as chaves completas e decifrar as tuplas.

O segundo problema do PVSS diz respeito ao algoritmo simétrico utilizado, que de acordo com os autores [13] do DepSpace, é o *Triple-DES* ($3DES$) [33]. Este algoritmo utiliza uma chave de 168 *bits* mas provê apenas 112 *bits* de segurança [7]. Além disso, para um algoritmo como este, só faria sentido se falar em resistência IND-CCA quando o modo de operação é autenticado como *Counter with CBC-MAC Mode* (CCM). Utilizando um modo de operação como *Cipher Block Chaining* (CBC) ou *Counter Mode* (CTR) a noção máxima de segurança é IND-CPA. Se porém, o modo de operação utilizado for *Electronic Code Block* (ECB), o algoritmo não atinge segurança semântica contra o CPA e é considerado inseguro [20]. No caso do DepSpace, o código implementado não especifica o modo a ser utilizado e portanto será utilizado o padrão do ambiente em que o sistema estiver sendo executado, que em alguns casos é o ECB.

4.2 Provendo Privacidade no DepSpace

Esta seção discute a proposta de incremento de segurança e funcionalidades no DepSpace através da aplicação dos esquemas criptográficos citados na Seção 2.3 ao protocolo. A ideia parte do mesmo princípio exposto na Seção 2.3.3, aplicando o conceito ao ambiente de coordenação por tuplas [13]. Fornecemos ainda uma arquitetura para compartilhamento de chaves utilizando o próprio espaço de tuplas e a segurança provida pelo esquema PVSS.

Como todas as garantias do sistema são baseadas no limite de até f falhas (ex.: camada de replicação), neste trabalho assumimos este mesmo limite. Portanto, a confidencialidade

do sistema, assim como suas demais propriedades, persiste enquanto até f servidores estiverem em falha, não havendo garantias caso um número maior de falhas ocorra.

4.2.1 Novas Classificações de Campos

A análise apresentada na Seção 4.1 mostra que, devido à forma como foi proposta a construção dos *fingerprints*, o DEPSPACE está sujeito a ataques simples. Portanto, faz-se necessário contornar este problema, evitando assim que os servidores manipulem dados em claro ou cifrados de forma fraca. Propõe-se, portanto, 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 $encryptCD(KeyCD_{shared}, f)$ (Seção 2.2.4);
- **Ordenável (OR)**, os quais serão cifrados utilizando um algoritmo simétrico de criptografia com preservação da relação de ordem $encryptOR(KeyOR_{shared}, f)$ (Seção 2.3.1);
- **Operável (OP)**, os quais serão cifrados utilizando um algoritmo completa ou parcialmente homomórfico $encryptOP(KeyOP_{public}, f)$ de acordo com a necessidade da aplicação (Seção 2.3.2).

Assim, para gerar o *fingerprint* $t_h = \langle h_1, \cdots, h_m \rangle$ de uma tupla $t = \langle f_1, \cdots, f_m \rangle$ com vetor de proteção $v_t = \langle v_1, \cdots, v_m \rangle$, a função $fingerprint(t, v_t) = t_h$ calcula cada $h_i = Enc_i(K_i, f_i)$, 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 $h_i = PR$. A função é apresentada na Figura 4.1.

O cliente então armazena a tupla no espaço seguindo o procedimento apresentado na seção 3.4.2. Com isso, continua garantido que um *template* $\overline{t}$ e a tupla t combinam se $\overline{t}_h$ combina com t_h. Então ao encontrarem um *fingerprint* t_h que combina com um *fingerprint*

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

Figura 4.1: Produção do *fingerprint*.

$\bar{t}_h$ de um *template* $\bar{t}$ utilizado para consulta, os servidores retornam seus *shares* cifrados. Estes serão decifrados e combinados pelo cliente para obter a tupla correspondente, a qual será verificada pelo mesmo.

Devido aos motivos citados na Seção 4.1, aconselha-se fortemente a preferência pelo uso da classificação *comparável determinístico* proposta ao invés da *comparável*, pois utilizando uma cifra com chave simétrica previamente compartilhada 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, 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 deduzido. 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 identificadores, endereços de *e-mail*, nomes de nós de processos, dentre outros.

Alguns algoritmos criptográficos utilizam modos de operação com Vetores de inicialização – *Initialization Vector (IV)* – randomizados como forma de prover criptografia probabilística. Este referido IV será então enviado como parâmetro da função de cifração juntamente com a chave e a mensagem na forma $Enc(k, m; IV)$. Estes algoritmos apresentam a 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 – *Pseudo Random Function (PRF)* – $F : (\mathcal{K} \times \mathcal{M}) \rightarrow \mathcal{R}$, onde $\mathcal{R}$ é um conjunto uniformemente distribuído. Sejam $k_1, k_2 \in \mathcal{K}$ chaves previamente compartilhadas, aplica-se F sobre a mensagem utilizando k_1, produzindo uma saída pseudo-aleatória $r = F(k_1, m)$, com $r \in \mathcal{R}$ e então utiliza-se r como IV da função de cifração utilizando a chave k_2, produzindo $c = Enc(k_2, m; r)$ seguindo o sugerido em [20]. 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 (Seção 2.2.4). Uma opção mais simples seria utilizar apenas uma função que gere um código de autenticação de mensagem, como HMAC-SHA256, já que este dado (*fingerprint*) não precisará ser decifrado futuramente.

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*.

Pelos motivos mostrados na Seção 2.3.1, sugere-se o uso do estado-da-arte deste tipo de algoritmo, chamado de Order Revealing Encryption (ORE), o qual foi apresentado por Lewi e Wu [52] e utilizado por Alves e Aranha [3]. Este algoritmo baseado em AES, ao invés de preservar a ordem, gera textos cifrados não determinísticos que não podem ser comparados diretamente, mas sim submetidos a uma função de comparação que retorna a relação entre eles. O acesso a esta função pode ser restringido, o que também contribui para um maior nível de segurança. Este algoritmo, por fornecer resposta também de igualdade, dispensaria o uso dos tipos de campos CO e CD, visto que pode ser aplicado também a textos e não apenas a números, tornando o sistema ainda mais seguro por seu caráter não determinístico. Porém, aqueles campos ainda podem ser utilizados por questões de desempenho.

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 como descrito na Seção 2.3.2. 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 uma classe de operações (como soma/subtração ou multiplicação/divisão), não se faz necessário o uso de uma cifra completamente homomórfica, uma cifra parcialmente homomórfica seria suficiente e não comprometeria de sobremaneira seu desempenho. Recomenda-se portanto o uso do algoritmo Paillier [59] para soma ou ElGamal [31] para multiplicação. Finalmente, como pode-se observar, ao ler uma tupla com um campo como este, deve-se considerar o valor do *fingerprint* e não o da tupla, pois esse é o campo que será atualizado caso a operação tenha sido aplicada em algum momento.

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 as chaves de decifração para os dados em questão. Além disso, os servidores não tomam conhecimento sequer do que está sendo consultado, possibilitando que uma busca segura seja realizada mesmo na presença de servidores não confiáveis.

4.2.2 Gerenciamento de Chaves

Gerenciamento de chaves não é uma tarefa trivial em sistemas distribuídos, e envolve desde o compartilhamento das chaves até a atualização das mesmas. O Algoritmo 1 apresenta um protocolo para compartilhamento de um par de chaves entre processos em um sistema distribuído, através de um espaço de tuplas. Algoritmo semelhante pode

Algoritmo 1 Algoritmo para compartilhamento de chaves.

```
1: void compartilharChaves() {
2:    my_key_private = gerar chave privada;
3:    my_key_public = gerar chave pública;
4:    t = ⟨"keys", my_key_public, my_key_private⟩ protegidos como ⟨PU, PU, PR⟩
5:    t̄ = ⟨"keys", *, *⟩ protegidos como ⟨PU, PR, PR⟩
6:    ⟨"keys", key'_pub, key'_priv⟩ = cas(t̄, t);
7:    if ⟨"keys", key'_pub, key'_priv⟩ != null then
8:       my_key_public = key'_pub
9:       my_key_private = key'_priv
10:   endif
11: }
```

ser usado para compartilhar apenas uma chave secreta. Nas soluções propostas neste trabalho, este algoritmo deve ser usado antes que os campos CD, OR e OP possam ser utilizados (note que este algoritmo utiliza apenas campos PU e PR).

A ideia por traz do algoritmo é que cada processo gere um par de chaves e tente disponibilizá-las no espaço para os outros processos. Para isso, utiliza a função *cas* com um *template* que combine com um par de chaves já disponível no espaço. Caso o retorno seja nulo, então estas foram as primeiras chaves a serem inseridas no espaço e o processo pode continuar a usá-las. Caso contrário, as chaves lidas do espaço passam a ser utilizadas. Como a chave privada fica em um campo PR, apenas os clientes que obtêm esta tupla do espaço conseguem acessá-la.

Note que devem-se ativar nestas tuplas os mecanismos de controle de acesso de forma que apenas os clientes autorizados, que fazem parte do grupo de coordenação, possam acessá-las. Por simplicidade, estas configurações de controle de acesso não são apresentadas nos algoritmos presentes neste trabalho. Pode-se notar que é possível o uso de diferentes chaves para diferentes campos, ou mesmo criar grupos que compartilham um mesmo conjunto de chaves. Finalmente, para atualizar esta chave ou o grupo de processos autorizados a acessá-la, basta inserir uma nova tupla no espaço com as configurações e dados apropriados. Para remover um processo do grupo, uma nova chave deve ser gerada (neste caso os *fingerprints* obtidos com a chave antiga devem ser atualizados). Por outro lado, a mesma chave pode ser usada para adicionar um processo no grupo, inserindo uma nova tupla com os mesmos valores, porém com a configuração apropriada de controle de acesso (incluindo o novo processo) e o novo processo só precisa ler a tupla do espaço.

4.2.3 Implementação

Utilizamos a implementação original do DepSpace, adicionando a ela algumas modificações que introduzem a nova classificação anteriormente apresentada para produção

do *fingerprint* e posteriores consultas e alterações de valores. Basicamente, as operações continuam a funcionar como em sua versão anterior e os seguintes algoritmos foram utilizados:

- Público (PU): não utiliza criptografia uma vez que o seu conteúdo original é usado como *fingerprint*.
- Comparável (CO): estes campos utilizam o algoritmo SHA-1, que gera um *hash* de 20 *bytes*. (legado da implementação original do DEPSPACE).
- Comparável Determinístico (CD): estes campos utilizam o algoritmo HMAC-SHA256 (*Hash-based Message Authentication Code with SHA-256*) fornecido pela biblioteca nativa javax.crypto. Foram utilizadas chaves secretas de 256 *bits* para gerar saídas pseudo-aleatórias de 256 *bits*.
- Ordenável (OR): para este tipo de campo foram utilizadas duas bibliotecas:
  - A biblioteca *jope* [62], uma implementação em Java do algoritmo OPE [17]. As consultas foram realizadas no servidor utilizando os métodos nativos de comparação $<$, $\leq$, $=$, $\geq$ e $>$ entre inteiros de precisão arbitrária (*BigInteger*).
  - A biblioteca FastORE [51], uma implementação em C do algoritmo ORE [52], integrada ao sistema via JNI [34]. A comparação é feita via função fornecida pelo algoritmo.

  Em ambas foram utilizadas chaves simétricas de 128 *bits* e para executar consultas de ordem. Implementamos no lado do cliente um detector que identifica a presença deste tipo de consulta, a manipula e cifra a mensagem de modo que a camada correspondente do lado dos servidores possa executar a consulta. Esta por sua vez foi ligeiramente alterada para realizar tais consultas ao invés de apenas igualdade.
- Operável (OP): estes campos utilizam a biblioteca *javallier* [4], que é uma implementação em Java do algoritmo de Paillier [59]. Para este algoritmo assimétrico utilizamos um par de chaves (pública e privada) de 3072 *bits*, de modo a obter um nível de segurança de 128 *bits* [3].
- Privado (PR): não utiliza criptografia uma vez que nenhuma informação é incluída no *fingerprint* (somente o símbolo PR).

4.2.4 Análise de Segurança da Proposta

Como citado na Seção 2.2.3, 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

Campo	Algoritmo	Resistência	Chaves	Nível de Segurança
PU	Não há criptografia	-	-	Inseguro
CO	SHA-1	Quebrado	-	Inseguro
OR	OPE	POPF-CCA	128 *bits*	128 *bits*
CD	HMAC-SHA256	IND-DCPA	256 *bits*	128 *bits*
OR	ORE	IND-OCPA	128 *bits*	128 *bits*
OP	Paillier	IND-CPA	3072 *bits*	128 *bits*
PR	Não há informações	-	-	Não se aplica
Tupla	PVSS+3DES	IND-CPA	168 *bits*	112 *bits*

Tabela 4.1: Nível de segurança dos campos no DepSpace

forte. Tomando isso em conta, apresentamos na Tabela 4.1 o nível de indistinguibilidade intrínseco a cifra utilizada em cada campo. Apresentamos na mesma tabela os tamanhos de chaves utilizados e o nível de segurança fornecido.

Os níveis IND-DCPA e IND-OCPA são ambos flexibilizações do nível IND-CPA, porém apesar de revelar igualdade e relação de ordem entre textos cifrados diferentes, o nível IND-OCPA só pôde ser alcançado graças ao não determinismo da cifra utilizada (ORE [52]). Além disso, a cifra impossibilita a inferência de dados a partir dos dados cifrados, sendo necessário uma função de comparação. Por estes motivos, considera-se o IND-OCPA como sendo um nível superior de segurança em relação ao IND-DCPA. Os algoritmos utilizados nos campos OR são, ambos, simétricos e baseados em AES, e por isso herdam o mesmo nível de segurança em relação ao tamanho da chave utilizada, guardadas as restrições já comentadas.

Cabe ressaltar que a composição HAMAC-SHA256 poderia fornecer até 256 *bits* de segurança contra colisões dependendo da qualidade da chave utilizada. Para isso, é necessário que esta tenha sido retirada aleatoriamente de um espaço uniformemente distribuído com pelo menos 2^{256} valores possíveis. Aplicar uma função de *hash* com uma saída como esta (256 *bits*) para tentar mascarar um espaço pequeno de chaves possíveis não adiantaria, pois como o algoritmo é determinístico, a chave fica vulnerável à pré-imagem, podendo o adversário calcular os *hashes* de todas as chaves possíveis e testá-las para uma mesma mensagem cuja saída HMAC-SHA256 seja conhecida. Admitindo esta possibilidade e pautados pela definição de segurança baseada em oráculo (Seção 2.1), em que o protocolo permite ao adversário enviar uma mensagem e obter o criptograma correspondente, estabelecemos o nível de segurança apenas nos 128 *bits* fornecidos pelo SHA256 contra colisões.

Para os campos operáveis, como os algoritmos aplicados são de chave pública, faz-se necessário o uso de chaves bem maiores (3072 *bits*) para fornecer os mesmos 128 *bits* dos

demais algoritmos [3].

Os campos privados dos *fingerprints* não contém informações referentes ao seu correspondente na tupla, e portanto, sua segurança está diretamente ligada ao nível de segurança da cifra utilizada para cifrar a tupla completa. Esta, como já comentado, é cifrada utilizando o esquema PVSS e o algoritmo de cifra simétrica 3DES, permanecendo segura sob as mesmas garantias de tolerância a falhas do esquema. Entretanto este algoritmo pode fornecer apenas 112 *bits* de segurança, um valor menor que os algoritmos sugeridos para os campos do *fingerprint*. Apesar de ainda ser aceitável na configuração que utiliza três chaves diferentes (*tree-key TDEA*) [8], sugere-se que o algoritmo 3DES seja substituído pelo AES-128 para prover os mesmos 128 *bits* de segurança dos algoritmos sugeridos. Recomenda-se ainda, a especificação de um modo de operação com autenticação como CCM para que esta cifra tenha resistência contra ataques de texto cifrado escolhido (IND-CCA).

4.2.5 Avaliação Experimental

Para demonstrar o uso das soluções propostas e melhor entender os custos envolvidos, executamos alguns experimentos com a implementação descrita acima.

Configuração dos Experimentos

Os testes foram realizados no ambiente do Emulab [75] configurado com 5 máquinas (2.4 GHz 64-bit Intel Xeon E5-2630v3 de 8 núcleos com 2 *threads* por núcleo, 64Gb de RAM e placas de rede de 1Gbps) em uma rede de 1Gbps. O ambiente de *software* utilizado foi sistema operacional Ubuntu 14 64-bit, *Java Virtual Machine* versão 1.8.0.121 e DepSpace versão 2.0 [16]. Para todos os experimentos, o sistema foi configurado com 4 réplicas hospedadas em diferentes maquinas para tolerar até uma réplica em falha, enquanto os clientes foram executados na máquina restante.

Para executar operações *out*, para cada classificação (PU, CO, PR, CD, OP, OR_OPE e OR_ORE), utilizamos tuplas com todos os campos definidos da mesma forma (ex: $\langle 1, 1, 1 \rangle$, com vetor de proteção $\langle OR\_ORE, OR\_ORE, OR\_ORE \rangle$). Por outro lado, os *templates* usados para as operações de *rdp* e *inp* foram configurados com apenas um campo definido enquanto todos os outros foram deixados como coringas (ex.: $\langle 1, *, * \rangle$, com vetor de proteção $\langle CD, CD, CD \rangle$, exceto pelas classificações PR e OP, as quais tiveram todos os campos configurados como coringas pois estas não possibilitam consultas.

Avaliamos a vazão (*throughput*) do sistema nos servidores e a latência percebida pelos clientes em todas as configurações. Para avaliar a latência, usamos um cliente para executar cada operação 1000 vezes, enquanto que para a vazão na tentativa de sobrecarregar os

servidores, cada cliente processou 1000 requisições, antes de enviá-las aos servidores, que mediram a vazão periodicamente a cada 100 requisições executadas. Este procedimento é necessário pois as operações criptográficas mais custosas são realizadas no lado do cliente. Apesar do DEPSPACE tolerar falhas, todos os valores de desempenho foram obtidos em execuções sem falhas.

Para a latência, ordenamos os valores obtidos e então trabalhamos com dois tipos de amostra: (a) selecionando o de posição $90 \cdot n/100$, onde n denota o número de operações realizadas. Ou seja, 90% dos valores foram menores ou iguais ao selecionado, chamaremos esta amostra de 90º percentil; e (b) calculamos a média de 90% das amostras descartando os 10% de maior variância. Já para a vazão, tomamos o valor máximo obtido para cada configuração. Em todas as operações foram avaliados o desempenho do processamento de tuplas com 3, 5 e 7 campos para cada classificação.

Para obter os custos relativos a cada processamento criptográfico envolvido nas operações (*fingerprint*, verificação, extração e comparação) executamos cada parte 10000 vezes para cada tipo de campo e calculamos a média descartando os 10% de maior variância.

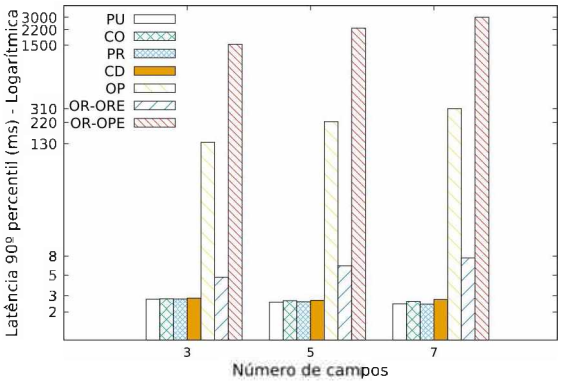
Resultados

Esta seção reporta os resultados obtidos nos experimentos. Primeiramente, as figuras 4.2(a) e 4.2(b) apresentam a latência (90º percentil e média) para operações *out*. Os gráficos são apresentados em escala logarítmica para facilitar a visualização. Como podemos ver, os três campos originais (PU, CO e PR) tem custos semelhantes, pois destes, o único que efetua alguma operação criptográfica é o comparável com a função SHA-1, a qual tem custo muito baixo.

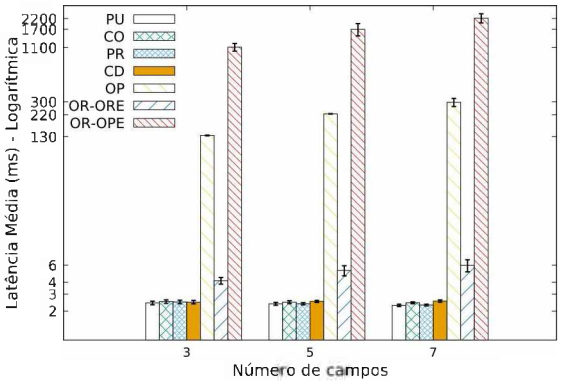
Os gráficos também nos mostram o campo CD com praticamente a mesma latência dos três campos originais do DEPSPACE, entre 2 e 3 *ms* para todos os números de campos. Um pequeno aumento foi apresentado nos campos OR_ORE, os quais permaneceram entre 5 e 7 *ms* na média com um amento gradativo para cada quantidade de campo.

O impacto começa a se tornar expressivo para os campos OP. Nestes campos, a latência se manteve entre 130 e 300 *ms* na média, com valores um pouco mais altos para o 90º percentil. Este aumento se deve ao tamanho da chave do algoritmo, 3072 *bits*. Para se ter uma ideia, em testes com 512 *bits* em ambiente similar o mesmo se manteve na casa dos 10 *ms* [36], entretanto este tamanho de chave não forneceria segurança suficiente.

Já no processamento dos campos OP_OPE a diferença se mostra muito mais expressiva. A latência varia de 1,1 a 2,2 *s* na média e de 1,5 a 3 *s* no 90º percentil, cerca de 1000x o custo dos campos mais baratos. Este custo alto muito provavelmente se deve ao tipo de função em que o algoritmo se baseia, chamadas de funções de distribuição hipergeométrica, as quais apresentam altos custos computacionais.

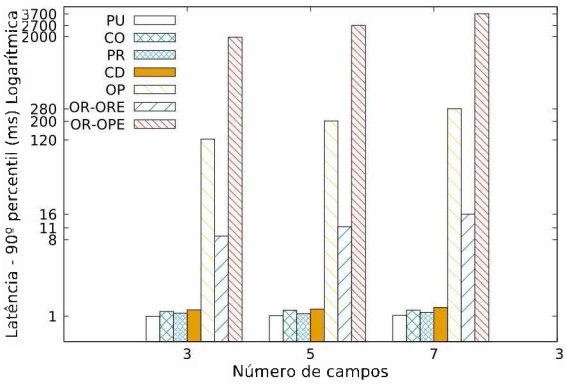


(a) Latência - 90º Percentil (*out*).

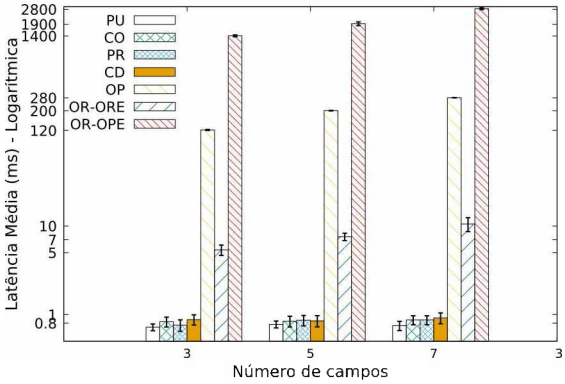


(b) Latência - Média (*out*).

Figura 4.2: Latência da inserção de tuplas (*out*).

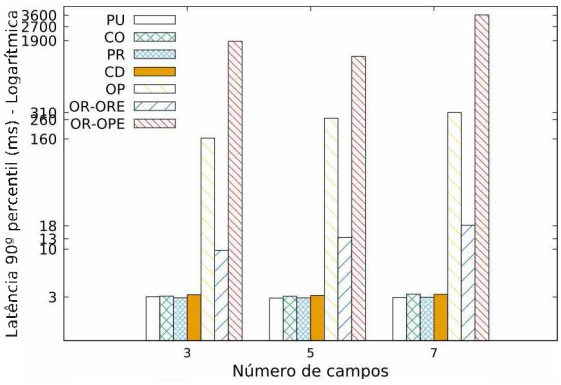


(a) Latência - 90º Percentil (*rdp*).

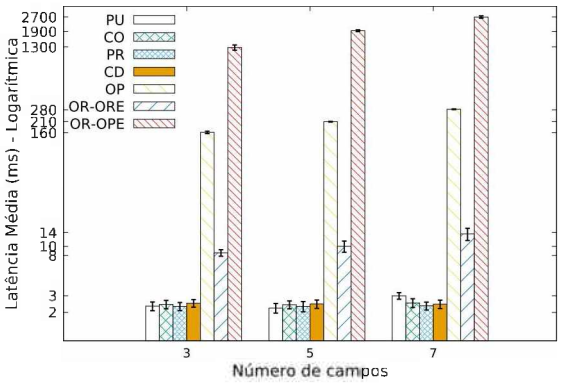


(b) Latência - Média (*rdp*).

Figura 4.3: Latência da leitura de tuplas (*rdp*).



(a) Latência - 90º Percentil (*inp*).



(b) Latência - Média (*inp*).

Figura 4.4: Latência da remoção de tuplas (*inp*).

Tabela 4.2: Custos médios ($\overline{md}$) e e o desvio padrão (σ) relativos ao processamento criptográfico de apenas um campo (σ) em milissegundos (ms).

	Fingerprint ($\overline{md}$ / σ)	Verificação ($\overline{md}$ / σ)	Extração ($\overline{md}$ / σ)	Busca/Comparação ($\overline{md}$ / σ)
PU	–	–	–	–
PR	–	–	–	–
CO	(0.003 / 0.0003)	(0.003 / 0.0003)	–	–
CD	(0.013 / 0.005)	(0.013 / 0.006)	–	–
OP	(32.879 / 0.071)	–	(13.308 / 0.025)	–
OR-ORE	(0.301 / 0.049)	(0.883 / 0.166)	–	(0.517 / 0.089)
OR-OPE	(287.453 / 42.604)	(320.748 / 67.061)	–	(0.002 / 0.0005)

Os custos referentes à criptografia para esta operação (*out*) são relacionados apenas à produção do *fingerprint* que é diferente para cada abordagem e a execução do esquema PVSS.

Comportamentos relativos similares entre os algoritmos podem ser visualizados para as operações *rdp* (Figuras 4.3(a) e 4.3(b)) e *inp* (Figuras 4.4(a) e 4.4(b)). Entretanto, é visível que na *rdp* os custos são significativamente menores para os campos com algoritmos mais baratos. Isso se deve a uma otimização do DepSpace para esta operação [13] e ao fato de que a operação *inp* além de ler, remove a tupla do espaço. Já para os algoritmos mais caros os custos criptográficos dominam a execução e a diferença entre as operações para o mesmo tipo de campo são pouco notadas.

Os custos com criptografia para estas operações (*rdp* e *inp*) são relacionados a: (1) o cliente precisa gerar o *fingerprint* $\overline{f}$ para o *template*; (2) (4) executar uma comparação nos servidores; verificar se $\overline{f}$ é válido para a tupla recebida; (3) extrair os valores dos campos operáveis do *fingerprint*, além da execução com PVSS.

A Tabela 4.2 apresenta os custos para cada etapa de criptografia. Como se pode notar, a maior parte dos custos com criptografia ocorrem no lado do cliente, isto é, os custos reportados nas colunas *Fingerprint*, Verificação e Extração. Na coluna Busca/Comparação, somente os campos OR têm processamentos adicionais executados pelos servidores, sendo que apenas o algoritmo ORE apresenta um custo significativo.

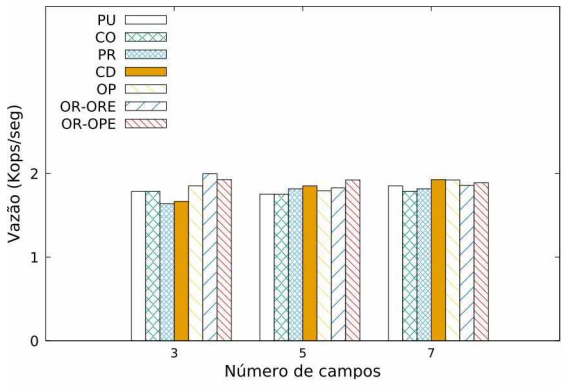
Podemos observar ainda que uma configuração com campos PU ou PR não apresenta custos pois nenhuma operação criptográfica é necessária para gerar seus respectivos *figerprints*. Nas configurações CO e CD os custos também são baixos. Como pode ser visto, apesar do processamento do *fingerprint* para os campos CD demandar maior tempo em relação aos campos CO (cerca de $4x$), este fato praticamente não impacta seu desempenho, pois estes algoritmos não dominam o tempo de execução e como pode ser visto nos

gráficos, se comportam de maneira similar aos campos PU e PR.

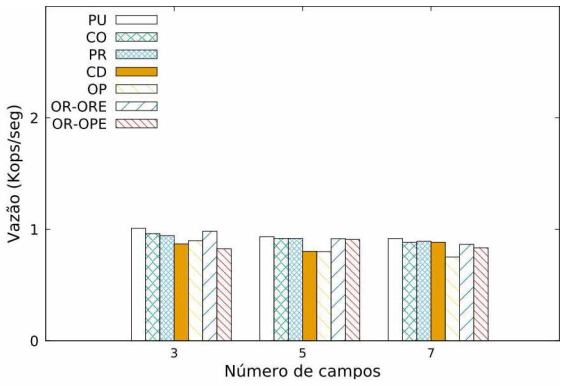
A configuração OP, cujo tempo na Tabela 4.2 é de cerca de $30k$ vezes o custo do campo CO, apresenta uma latência de 100 a $300x$ em relação ao mesmo campo em todos os gráficos.

Nos campos OR é possível perceber uma grande diferença entre os dois algoritmos testados. Na configuração utilizando o algoritmo ORE, o processamento do *fingerprint* leva em média cerca $100x$ o tempo médio do CD. Entretanto, como pode ser visto nos gráficos, a latência deste campo em relação aos mais baratos é de apenas $2x$ para todas as operações.

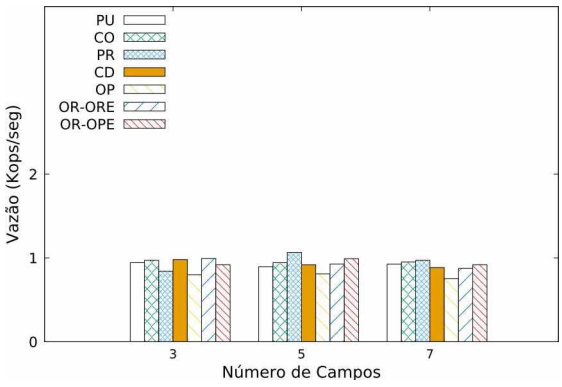
Por outro lado, a configuração OR utilizando o algoritmo OPE apresentou um desempenho proibitivo visto que levou um tempo extremamente maior para processar seu *fingerprint*, cerca de $100k$ vezes o tempo do campo CO. O impacto pode ser visto nos gráficos em que a latência para todas as operações é de mais de $1000x$ a do campo CO. Cerca de $600ms$ são gastos com processamento e verificação de apenas um campo de



(a) Vazão da inserção de tuplas (*out*).



(b) Vazão da leitura de tuplas (*rdp*).



(c) Vazão da remoção de tuplas (*inp*).

Figura 4.5: Vazão (*throughput*) de inserção (*out*), leitura (*rdp*) e remoção (*inp*) de tuplas do espaço.

cada requisição. Com um pequeno aumento das requisições a diferença já é facilmente perceptível pelo usuário.

Além disso, o maior custo relacionado à execução do esquema PVSS também ocorre do lado do cliente [13]. Isto é importante porque mostra que é possível ter um espaço de tuplas que garante privacidade das informações que armazena e ao mesmo tempo ser escalável. Consequentemente, apesar destes custos impactarem a latência, não é esperado que tenham impacto significativo na vazão do sistema.

Tentando investigar este aspecto, a Figura 4.5 apresenta a vazão (*throughput*) apresentada nos servidores para cada operação e configuração. A vazão é similar para todas as configurações, mesmo para os campos OR com algoritmo ORE, que utilizam uma função de comparação. Não obstante, o *throughput* das inserções (Figura 4.5(a)) de tuplas no espaço é maior que o de leitura (Figura 4.5(b) ou remoção (Figura 4.5(c)) do espaço. De fato, na execução de uma leitura ou remoção, os servidores precisam efetuar as buscas para selecionar uma tupla, enquanto nenhum processamento adicional é executado nos servidores para operações de inserção (Seção 3.4.2).

Um outro importante aspecto observado nos experimentos é que o número de campos em uma tupla não impacta significativamente o desempenho do sistema. De fato, para todas as configurações e operações, o desempenho para 3, 5 e 7 campos são similares.

Além disso, a Tabela 4.3 mostra a quantidade de informação (em *bytes*) que necessita ser trocada entre clientes e servidores em uma requisição/resposta para cada configuração. O tamanho da requisição e o entendimento do seu impacto nos custos de comunicação é importante pois estes dados são trafegados pelos protocolos de replicação que têm complexidade de $O(n^2)$ mensagens [13]. O tamanho da resposta também é importante pois vários servidores enviam respostas ao cliente resultando em uma comunicação de n pra 1.

Como pode ser visto, apenas o campo OP possui grande diferença na quantidade de

Tabela 4.3: Quantidade de dados (em *bytes*) transmitidos em uma requisição/resposta para cada configuração e operação.

	OUT (requisição / resposta)			RDP/INP (requisição / resposta)		
	3 campos	5 campos	7 campos	3 campos	5 campos	7 campos
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-ORE	(890 / 133)	(954 / 133)	(1018 / 133)	(320 / 1092)	(342 / 1198)	(364 / 1296)
OR-OPE	(825 / 133)	(866 / 133)	(916 / 133)	(297 / 1026)	(319 / 1081)	(341 / 1124)

dados trafegados em relação aos demais algoritmos. Isto se dá porque este é o único algoritmo de chave pública utilizado na proposta. Este tipo de algoritmo além de utilizar chaves maiores, produz criptogramas maiores para atingir os mesmos níveis de segurança que os demais da proposta.

Note que estes resultados são ligeiramente diferentes daqueles apresentado no artigo original do DEPSPACE [13], pois nestes experimentos não utilizamos a otimização no *rdp* proposta pelos autores, a qual permite ignorar os protocolos de ordenação da RME para sequências de operações sem alteração de estado.

4.3 Discussão

Esta seção apresenta questões importantes sobre alguns aspectos da solução proposta.

4.3.1 (Im)possibilidade de Combinações de Campos

A implementação proposta não permite que um campo assuma mais que um tipo. Entretanto, alguns tipos podem ser vistos como uma combinação de outros tipos pois provêm funcionalidades equivalentes. Abaixo apresentamos alguns exemplos:

- OR também provê buscas por igualdade, dispensado o uso de CD ou CO.
- CO e CD provêm a mesma funcionalidade, porém com níveis de segurança diferentes
- PU provê todas as funcionalidades, mas não é seguro.

Por outro lado, as combinações apresentadas abaixo não são possíveis:

- campos OP não são determinísticos nem provêm métodos para realizarem comparações. Portanto não podem prover as funcionalidades de CD ou CO, tampouco OR.
- OP são os únicos campos seguros que possibilitam computação.
- OR são os únicos campos seguros que possibilitam ordenação.
- PR apresentam o melhor nível de segurança, mas não provê qualquer funcionalidade.

4.3.2 Replicação Máquina de Estados vs. Campos Operáveis

Campos operáveis podem ser alterados através da execução de computação nos servidores. Estas mudanças ocorrem no *fingerprint* e não na tupla, a qual está cifrada sob uma cifra padrão (3DES) como uma única peça de dados. Esta abordagem leva a duas questões que precisam de atenção:

- Primeiramente, quando decifrada após uma operação de leitura/remoção, o valor do campo OP do *fingerprint* pode não coincidir com o campo correspondente da tupla. Consequentemente, durante a verificação da validade do *fingerprint* para a tupla recebida (Seção 3.4.2), campos OP são sempre considerados válidos. Além disso, o valor do campo correspondente da tupla deve ser desconsiderado e sobrescrito pelo valor atualizado do campo do *fingerprint*.

- Por último e mais crítico, durante a execução dos protocolos de Replicação Máquina de Estados (RME) no cliente [64, 23], as respostas recebidas dos servidores podem ser diferentes pois os campos OP usam um algoritmo não determinístico. De fato, os criptogramas resultantes de alguma computação nos servidores podem diferir entre si, apesar de se referirem ao mesmo valor. Estes aspectos impactam nos protocolos de RME, os quais requerem $f + 1$ respostas idênticas para terminar retornando alguma destas réplicas. Para contornar este problema, é necessário decifrar os campos operáveis antes de contar o número de réplicas recebidas.

4.3.3 Comparação entre Abordagens de Espaços de Tuplas

Como comentado na Seção 3.4, existem vários sistemas desenvolvidos para prover propriedades de segurança e/ou tolerância a falhas no modelo de coordenação por espaço de tuplas. Como pode ser visto na Tabela 4.4, algumas destas propostas implementam apenas mecanismos de replicação [6, 76], enquanto outros usam apenas técnicas de controle de acesso [28, 22, 73]. Outros sistemas prevêm algum nível de tolerância a falhas através do conceito de transações [70, 41, 48]. O sistema DepSpace se destaca por prover

Tabela 4.4: Comparação entre abordagens de espaços de tuplas.

	Replicação	Transações	Controle de Acesso	Confidencialidade	Permite operações e busca sobre dados cifrados
TSpaces [70]	✗	√	✗	✗	✗
JavaSpaces [48]	✗	√	✗	✗	✗
GigaSpaces [41]	✗	√	✗	✗	✗
FT-Linda [6]	√	✗	✗	✗	✗
Parallel-Linda [76]	√	✗	✗	✗	✗
SecSpaces [22]	✗	✗	√	✗	✗
KLAIM [28]	✗	✗	√	✗	✗
SECOS [73]	✗	✗	√	✗	✗
DepSpace [13]	√	✗	√	√	✗
DepSpace +soluções propostas	√	✗	√	√	√

replicação, controle de acesso e algum nível de confidencialidade que, como já discutido aqui, é bastante limitado por considerar apenas ataques mais simples. Nosso trabalho vai um pouco mais além, elevando o nível de segurança do sistema e possibilitando ainda operações e buscas sobre os dados cifrados.

4.4 Conclusões

Apresentamos neste capítulo nossa proposta de extensão ao sistema de coordenação DepSpace. Estas propostas foram pautadas pela análise de segurança e funcionalidades apresentada no início do capítulo e embasadas nas primitivas de computação em dados cifrados fornecida no Capítulo 2, aplicadas ao ambiente de coordenação por espaço de tuplas descrito no Capítulo 3.

Através da inserção de novas classificações de campos para os *fingerprints* do DepSpace (CD, OR e OP), eliminamos a necessidade do uso de campos vulneráveis como PU e CO, além de adicionar novas funcionalidades às buscas e atualizações de dados no sistema de modo seguro. Como as novas classificações de campos envolvem o uso de esquemas criptográficos que necessitam de compartilhamento de chaves, utilizamos as funcionalidades já presentes no DepSpace para prover um protocolo de de gerenciamento de chaves. Apresentamos ainda os dados de desempenho obtidos em experimentos utilizando todos os campos implementados no DepSpace,

O próximo capítulo visa a utilização deste sistema proposto para aplicações práticas.

Capítulo 5

Segurança e Privacidade em Coordenação Distribuída Extensível

Este capítulo mostra como as melhorias propostas para o DepSpace podem ser usadas no desenvolvimento de mecanismos e protocolos para coordenação distribuída extensível.

5.1 Coordenação Distribuída Extensível (CDE)

Apesar de espaços de tuplas proverem as funcionalidades necessárias para coordenação, um estudo recente mostrou que o emprego de protocolos e arquiteturas de Coordenação Distribuída Extensível (CDE) é fundamental para o desempenho do sistema [30]. A ideia principal das extensões é permitir que os servidores, que mantém a infraestrutura de coordenação, acessem e processem as informações de coordenação. Os principais benefícios de se utilizar CDE são os seguintes: o número de passos de comunicações (normalmente, chamadas de procedimento remoto – RPC) necessárias para processar a coordenação é reduzido e evita-se reprocessamento de requisições quando o sistema passa por acessos concorrentes, pois é garantido que a requisição de coordenação sempre é processada utilizando o estado atualizado do sistema.

Como exemplos de mecanismos que podem se beneficiar de extensões, podemos citar os contadores compartilhados e as filas distribuídas que podem ser usados em *data centers* para coordenação e sincronização de processos [30]. No modelo tradicional de coordenação, para atualizar um contador um cliente necessita fazer a leitura do mesmo no servidor, atualizá-lo localmente e enviar o contador atualizado de volta ao servidor. Entretanto, se concorrentemente outros clientes também leram o contador com o mesmo valor, somente o primeiro a enviar o valor atualizado terá sucesso em sua operação. Os demais clientes deverão ler o contador novamente repetindo a operação até conseguirem sucesso. O

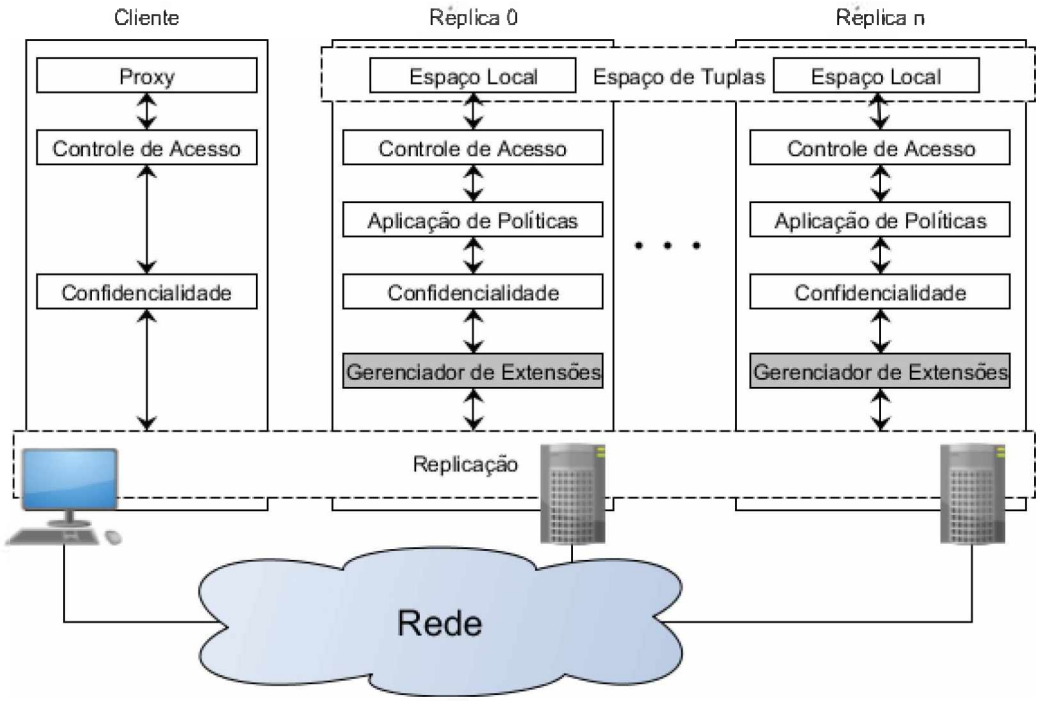


Figura 5.1: Camadas do DepSpace Extensível (Adaptado de [30]).

problema piora com o crescimento do número de clientes, diminuindo o desempenho do sistema.

Através da utilização de coordenação extensível, a operação de incremento do contador pode ser executada diretamente no servidor ao invés de ler o contador e atualizá-lo nos clientes, eliminando a necessidade de reprocessamentos devido a acessos concorrentes. Pensamento semelhante pode ser aplicado às filas distribuídas, de modo a garantir a ordem de inserção e recuperação dos elementos. As seções 5.2.1 e 5.2.2 abordarão estes mecanismos com mais detalhes, comentando suas respectivas versões em coordenação extensível.

Quanto às garantias de *liveness* do sistema para processos concorrentes, o modelo tradicional garante o progresso de um processo apenas em execução isolada para um tempo longo o suficiente (*obstruction-freedom* [43]). Já utilizando CDE, é garantido que qualquer operação termina se o processo que a invocou não falhar (*wait-freedon* [42]). Portanto com CDE tem-se garantias mais fortes de que todas as operações serão executadas.

5.2 Segurança e Privacidade em CDE

A Figura 5.1 apresenta as camadas do DepSpace, com uma camada adicional para gerenciamento de extensões [30], que não fornece segurança e privacidade adequadas, uma vez que dados em claro são acessados pelos servidores. Conforme já comentado,

através do uso de campos OP e OR, é possível que por meio das extensões nesta camada, servidores realizem computações sobre dados cifrados. Note que, devido aos problemas expostos, extensões seguras não podem ser implementadas sem a utilização destes campos.

Antes de uma extensão ser instalada, o gerenciador de extensões realiza uma série de verificações para determinar se este passo é seguro (ex.: uma extensão não pode conter laços infinitos de repetição ou recursividade). Depois que uma extensão é instalada, a mesma passa a interceptar as operações antes de serem processadas pelas camadas superiores e podem modificar o comportamento do sistema. No entanto, as mesmas devem apresentar um comportamento determinístico para manter a consistência no estado das réplicas e também são executadas de forma controlada, evitando por exemplo um consumo significativo de recursos (CPU), o acesso ao sistema de arquivos e a abertura de conexões [30].

As seções seguintes apresentam protocolos para implementação de dois importantes mecanismos de coordenação distribuída [30], que são os contadores compartilhados e as filas distribuídas. Por exemplo, estes mecanismos podem ser usados em *data centers* para coordenação e sincronização de processos. Para cada um dos mecanismos, são apresentadas e comparadas soluções que utilizam tanto o modelo tradicional de coordenação quanto o modelo extensível.

5.2.1 Contador Compartilhado

Para implementar um contador compartilhado, os processos precisam ler e atualizar um contador armazenado em uma tupla. O Algoritmo 2 apresenta a solução usando o modelo tradicional de coordenação. Neste algoritmo, para incrementar o contador, um processo (cliente) precisa primeiro ler o valor armazenado no espaço (linha 4) para então incrementá-lo (linha 5) e armazená-lo de volta no espaço substituindo o valor antigo (linha 6). Neste modelo, caso haja concorrência no acesso ao contador, um processo pode não conseguir atualizar seu valor devido ao fato de outro processo já ter realizado esta atualização. Neste caso, a função *replace* (linha 6) retorna nulo e todo o procedimento precisa ser reiniciado. Note que o valor do contador é armazenado em um campo CD, pois a função *replace* usa o valor antigo para localizar a tupla a ser substituída.

Através da utilização de coordenação extensível, o Algoritmo 3 diminui os passos de comunicação necessários para realizar a operação de incremento (apenas um *rdp* ao invés de um *rdp* e um *replace* do Algoritmo 2) e elimina a necessidade de reprocessamentos devido a acessos concorrentes. Neste caso, uma extensão (linhas 6-13) é instalada nos servidores e quando uma operação *rdp* é recebida, o servidor incrementa o valor do contador (para isso este campo é configurado como OP) e retorna o valor já atualizado.

Algoritmo 2 Contador compartilhado tradicional.

```
1: int incrementar() {
2:   while true do
3:     t̄= ⟨cont, ∗⟩ protegidos como ⟨PU, CD⟩;        { // define o template.}
4:     ⟨cont, c⟩ = rdp (t̄);                          { // lê o valor do contador}
5:     cnt_novo = ⟨cont, c + 1⟩ protegidos como ⟨PU, CD⟩;   { // define o novo valor}
6:     ⟨cont, c⟩ = replace(⟨cont, c⟩,cnt_novo);      { // tenta atualizar o contador}
7:     if ⟨cont, c⟩ != null then   { //termina em caso de sucesso, senão tenta novamente}
8:       return c + 1;
9:     end if
10:  end while
11: }
```

Nestes algoritmos, o prefixo "local" (linha 8) é utilizado para indicar um operação local nos servidores, que não envolve a troca de mensagens (execuções de RPCs).

Algoritmo 3 Contador compartilhado usando coordenação extensível.

Cliente:

```
1: int incrementar() {
2:   t̄= ⟨cont, 1⟩ protegidos como ⟨PU, OP⟩;        { // define o template.}
3:   ⟨cont, c⟩ = rdp (t̄);                          { // lê o valor já atualizado do contador}
4:   return c;
5: }
```

Extensão nos servidores:

```
6: Tuple rdp(⟨cont, cif_recebido⟩) {
7:   default = ⟨cont, ∗⟩ protegidos como ⟨PU, OP⟩;
8:   ⟨cont, cif_atual⟩ = local.rdp(default);   { //busca localmente o valor atual do contador}
9:   cif_novo = soma(cif_atual, cif_recebido)   { //operação possível por ser campo OP}
10:  cnt_novo = ⟨cont, cif_novo⟩ protegidos como ⟨PU, OP⟩;
11:  local.replace(⟨cont, cif_atual⟩, cnt_novo);
12:  return cnt_novo;
13: }
```

5.2.2 Fila Distribuída

Para implementar uma fila distribuída através de um espaço de tuplas, os processos precisam inserir tuplas no espaço representando os elementos da fila. Cada elemento contém um identificador que será usado para organizar os elementos em ordem na fila. Neste trabalho, utilizamos o tempo da criação dos elementos para esta finalidade. Posteriormente, para remover o primeiro elemento da fila, um processo precisa remover a tupla com o menor tempo de criação.

O Algoritmo 4 apresenta a solução que utiliza o modelo tradicional. Para inserir um elemento na fila basta inserir a tupla representando este elemento no espaço. No entanto, na remoção primeiramente todas as tuplas com elementos precisam ser lidas pelo processo (linha 8) que as organiza segundo o tempo de criação (linha 9) para então tentar remover o primeiro elemento da fila (linha 11). Esta função pode retornar nulo devido ao acesso concorrente de outro processo que já removeu o elemento em questão. Desta forma, aumentar a concorrência também aumenta a necessidade de reprocessamentos, em um comportamento similar ao que ocorre com o contador compartilhado previamente apresentado.

Finalmente, note que os dados do elemento são armazenados em campo privado (PR) e somente clientes autorizados podem acessar esta informação. Já o identificador do elemento é armazenado em campos CD, pois a função *inp* usa este campo para remover a cabeça da fila (linha 11).

Algoritmo 4 Fila distribuída tradicional.

```
 1: void add(ElementoId eid, byte[] dados) {
 2:   t = ⟨fila, eid, dados⟩ protegidos como ⟨PU, CD, PR⟩
 3:   out(t);
 4: }

 5: byte[] remove() {
 6:   while true do
 7:     t̄= ⟨fila, *, *⟩ protegidos como ⟨PU, CD, PR⟩;      { // define o template.}
 8:     Tuple[] tuples = rdAll (t̄);                        { // lê os elementos}
 9:     Ordenar as tuplas em tuples pelos seus ids (segundo campo)
10:     for all t in tuples do
11:       ⟨fila, id, dados⟩ = inp(t);      { //tenta remover o primeiro elemento da fila}
12:       if ⟨fila, id, dados⟩ != null then   { //termina caso conseguir remover a tupla}
13:         return dados;
14:       end if
15:     end for
16:   end while
17: }
```

O Algoritmo 5 apresenta a solução usando coordenação extensível. Assim como no contador, esta solução diminui os passos de comunicação e elimina a necessidade de reprocessamentos. A inserção de um elemento funciona como no modelo anterior, já na remoção uma extensão é instalada nos servidores para interceptar operações *inp* e remover o primeiro elemento da fila, retornando-o. Para isso, os campos com os identificadores dos elementos devem ser configurados como OR. Ao receber uma solicitação de remoção, o próprio servidor ordena localmente as tuplas com base neste campo, retirando a de menor *id*.

Algoritmo 5 Fila distribuída usando coordenação extensível.

Cliente:

```
1: void add(ElementoId eid, byte[] dados) {
2:    t = ⟨fila, eid, dados⟩ protegidos como ⟨PU, OR, PR⟩
3:    out(t);
4: }

5: byte[] remove() {
6:    t̄= ⟨fila, *, *⟩ protegidos como ⟨PU, OR, PR⟩
7:    ⟨fila, id, dados⟩ = inp(t̄);
8:    return dados;
9: }
```

Extensão nos servidores:

```
10: Tuple inp(⟨fila, *, *⟩) {
11:    default = ⟨fila, *, *⟩ protegidos como ⟨PU, OR, PR⟩;
12:    Tuple[] tuples = local.rdAll(default);        { // lê localmente os elementos}
13:    head = tupla em tuples com menor id    { //comparação possível por ser campo OR}
14:    local.inp(head);
15:    return head;
16: }
```

5.3 Avaliação Experimental

Visando analisar o desempenho dos mecanismos propostos para coordenação distribuída, os algoritmos foram implementados no DepSpace e alguns experimentos foram realizados no Emulab [75]. O principal objetivo destes experimentos é analisar as diferenças entre as abordagens e demonstrar o quanto as extensões podem melhorar o desempenho do sistema em um cenário considerando segurança e privacidade, seguindo o mesmo padrão apresentado em um cenário sem segurança [30].

Configuração dos Experimentos. O ambiente para os experimentos foi constituído por 6 máquinas (2.4 GHz 64-bit Intel Xeon E5-2630v3 de 8 núcleos com 2 *threads* por núcleo, 64GB de RAM e interface de rede de 1Gbps) conectadas a um *switch* de 1Gbps. O ambiente de *software* utilizado foi o sistema operacional Ubuntu 14 64-*bit*, *Java Virtual Machine* de 64 *bits* versão 1.8.0_131 e o Extensible DepSpace (EDS) [30], uma versão do DepSpace voltada para coordenação extensível. O sistema foi configurado com 4 servidores hospedados em máquinas diferentes para tolerar a falha de até um destes. Cada servidor foi executado em uma máquina separada, enquanto que os clientes foram distribuídos uniformemente nas outras duas máquinas.

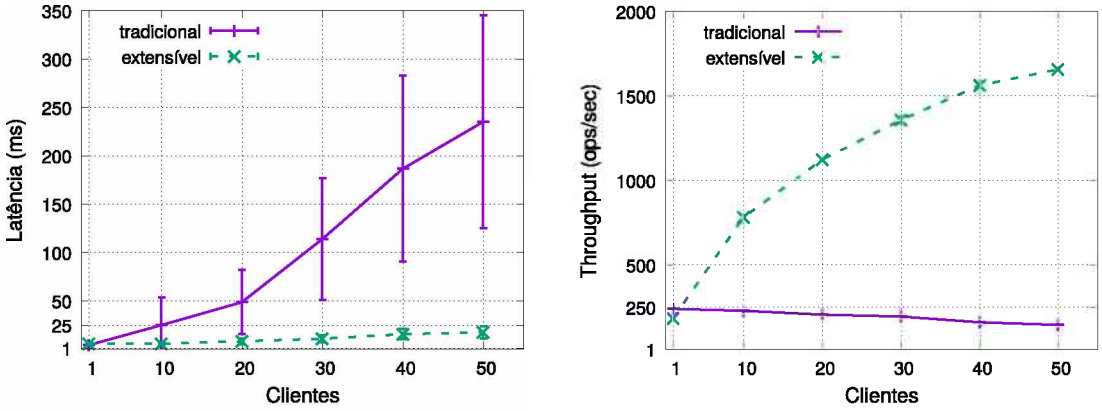
Foram utilizados para cada campo os seguintes algoritmos:

- Público (PU): não utiliza criptografia uma vez que o seu conteúdo original é usado como *fingerprint*.
- Comparável Determinístico (CD): estes campos utilizam o algoritmo HMAC-SHA256 (*Hash-based Message Authentication Code with SHA-256*) fornecido pela biblioteca nativa javax.crypto. Foram utilizadas chaves secretas de 256 *bits* para gerar saídas pseudo-aleatórias de 256 *bits*.
- Ordenável (OR): para este tipo de campo foi utilizada a biblioteca FastORE [51], uma implementação em C do algoritmo ORE [52], integrada ao sistema via JNI [34], com chaves simétricas de 128 *bits*. A comparação é feita via função fornecida pelo algoritmo.
- Operável (OP): estes campos utilizam a biblioteca *javallier* [4], que é uma implementação em Java do algoritmo de Paillier [59]. Para este algoritmo assimétrico utilizamos um par de chaves (pública e privada) de 3072 *bits*.
- Privado (PR): não utiliza criptografia uma vez que nenhuma informação é incluída no *fingerprint* (somente o símbolo PR).

Para verificar o desempenho das diferentes abordagens, variamos a quantidade de clientes e medimos a vazão (*throughput*) e a latência apresentadas pelo sistema. A vazão foi medida nos servidores a cada 1000 requisições processadas, enquanto que a latência foi medida nos clientes em uma execução com 1000 repetições. Os mesmos métodos do Capítulo 4 foram aplicados para cálculos das médias. Os gráficos desta seção apresentam os valores médios obtidos para estas métricas. Nos experimentos com a fila distribuída, cada cliente primeiramente adicionava um dado (como elemento da fila, usamos um vetor com zero *bytes* para avaliar apenas os custos relacionados com as iterações no protocolo) e depois removia o primeiro elemento da fila, garantindo que a mesma nunca estivesse vazia no momento da remoção.

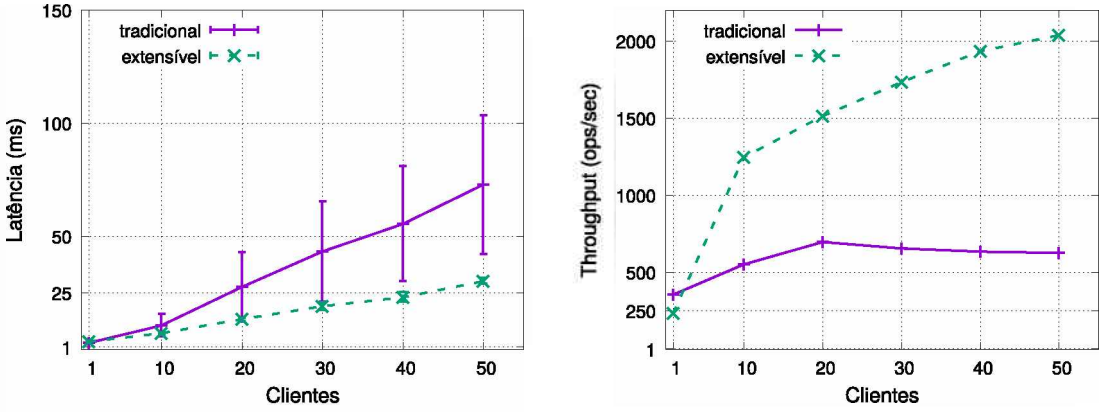
5.3.1 Resultados e Análises

A Figura 5.2 apresenta os valores para o desempenho dos protocolos para o contador compartilhado, enquanto que a Figura 5.3 apresenta os valores para a fila distribuída. De uma forma geral, as soluções que permitem processamento nos servidores (extensíveis) apresentam um desempenho significativamente superior ao das soluções tradicionais. De fato, conforme o número de clientes aumenta, a concorrência para acesso aos dados compartilhados aumenta e a quantidade de reprocessamentos (devido a falhas no incremento do contador ou na remoção do início da fila) também se torna maior. Este comportamento acaba impactando negativamente o desempenho do sistema em sua forma tradicional.



(a) Latência (b) *Throughput*

Figura 5.2: Latência e vazão (*throughput*) do contador compartilhado.



(a) Latência (b) *Throughput*

Figura 5.3: Latência e vazão (*throughput*) da fila distribuída.

Entretanto, na versão extensível o desempenho é significativamente melhor utilizando processamentos nos servidores mesmo que para isso precise realizar computações criptográficas mais custosas (campos OP ou OR).

Os experimentos mostraram que o desvio padrão para a latência das soluções tradicionais é muito grande, o que é explicado pelo fato de algumas operações não precisarem de reprocessamentos enquanto que outras podem repetir várias vezes até conseguirem realizar as computações desejadas (incrementar o contador ou remover o início da lista). Este comportamento corrobora com as garantias de *liveness* citadas anteriormente, fornecendo melhores garantias de execução das operações concorrentes. Além disso, a diferença no desempenho é maior no contador compartilhado que na fila distribuída, pois o protocolo para inserir um elemento na fila é exatamente o mesmo para ambas abordagens e portanto,

este mecanismo se beneficia das soluções propostas apenas na retirada de elementos.

Com base nos gráficos, podemos concluir que em média, a latência foi reduzida em cerca de 90% para o contador e 60% para a fila. Já quanto a vazão, podemos observar no caso extensível valores quase 9 vezes maiores para o contador e mais que 3 vezes maiores para a fila em relação às abordagens tradicionais. Finalmente, vale destacar que em testes semelhantes mas sem as opções de segurança e privacidade, a vazão do sistema com extensões foi cerca de 10x maior que o observado com estas opções habilitadas.

5.4 Conclusões

Neste capítulo demonstramos o poder computacional das soluções propostas, aplicando-as ao modelo de coordenação extensível. Experimentos foram apresentados com vistas de comprovar este resultado. Pudemos concluir que, apesar do impacto gerado pelo uso de algoritmos criptográficos demonstrado no Capítulo 4, os mecanismos apresentados tiveram uma significativa melhora de desempenho se comparados aos seus correspondentes no modelo tradicional de coordenação.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste capítulo apresentamos nossa conclusão na forma de uma breve visão do trabalho realizado, juntamente com uma revisão dos objetivos demonstrando como cada um foi alcançado. Apresentamos também nossas perspectivas de trabalhos futuros.

6.1 Visão Geral do Trabalho

Este trabalho apresentou uma proposta de incremento de segurança ao modelo de coordenação por espaço de tuplas, utilizando para tal o sistema DEPSPACE. Primeiramente, uma breve explanação sobre os aspectos de segurança da informação foi apresentada no Capítulo 2, juntamente com os algoritmos considerados como o "estado da arte" para computação privativa e os principais trabalhos que os utilizam.

Uma visão geral sobre os sistemas de coordenação distribuída foi apresentada no Capítulo 3 com ênfase nos espaços de tuplas. No mesmo capítulo foram mostrados os principais trabalhos que visam agregar tolerância a falhas e propriedades de segurança a este modelo.

O Capítulo 4, por sua vez, trouxe nossas propostas, utilizando os algoritmos apresentados no Capítulo 2 para prover segurança ao modelo de coordenação apresentado no Capítulo 3. O capítulo, que foi iniciado com uma análise de segurança do sistema em questão, apresentou ao final os resultados experimentais, quantificando o impacto de desempenho causado pelos algoritmos criptográficos. Uma análise de segurança do sistema proposto também foi apresentada, qualificando os níveis de segurança alcançados.

Pautados pelas propostas de coordenação extensível apresentada em [30], no Capítulo 5 apresentamos exemplos de mecanismos que puderam se beneficiar dos algoritmos propostos através da busca e operação em dados cifrados. Experimentos realizados demonstraram que mesmo em face custo dos algoritmos criptográficos, foi possível obter uma significativa melhora de desempenho nos mecanismos de coordenação.

Como conclusão geral do trabalho, percebemos que através desta proposta, foi possível incrementar tanto a segurança do sistema quanto suas funcionalidades, tendo ainda uma melhora de desempenho quando aplicada a mecanismos reais. Com isso, trazemos a aplicação de conceitos de segurança da informação para mais próximo do desenvolvimento das aplicações práticas, estreitando o largo abismo que costuma separar estes dois mundos.

6.1.1 Revisão dos Objetivos

Revisamos aqui os objetivos apresentados na Seção 1.2 e de que forma alcançamos cada um deles. Iniciando pelos objetivos específicos temos que:

"Estudar os conceitos relacionados com Sistemas Distribuídos (SD) focando em coordenação de processos;"

Foi possível no decorrer deste estudo, tomar conhecimento destes conceitos e problemas através do estudo das principais fontes da área, onde foi possível o contato com o sistema DEPSPACE. Seus principais pontos foram expostos no Capítulo 3.

"Estudar os conceitos de Segurança da Informação e os diversos protocolos de criptografia, buscando algoritmos apropriados ao cenário de comunicação por tuplas;"

Da mesma forma, os principais algoritmos foram estudados e testados tendo como fruto a seleção dos que foram utilizados neste trabalho, os quais foram relacionados no Capítulo 2.

"Analisar a segurança do sistema de coordenação DEPSPACE *[13] e propor extensões ao sistema visando torná-lo mais seguro;"*

Com base nos conceitos estudados, foi possível realizar esta análise e proposta, como mostrado no Capítulo 4.

"Implementar as soluções propostas e analisar o desempenho e nível de segurança do sistema resultante;"

Após a seleção dos algoritmos, buscamos e testamos bibliotecas que os implementam, apresentando sua análise de segurança e experimentos que mediram o desempenho dos campos propostos dando-nos uma boa noção do impacto dos algoritmos criptográficos.

"Aplicar as soluções propostas no desenvolvimento de protocolos de coordenação distribuída extensível, que façam uso de toda sua potencialidade."

Mecanismos de contador compartilhado e fila distribuída foram apresentados e suas versões extensíveis utilizando as soluções de privacidade foram comparadas com seus modelos tradicionais, demonstrando ganhos expressivos.

Quanto ao objetivo geral de *"projetar e desenvolver protocolos para prover segurança, principalmente privacidade, em dados compartilhados através de espaço de tuplas"*:

Acreditamos que o objetivo geral foi alcançado pela soma dos motivos citados acima. Corroborando com isso, uma parte das contribuições acima listadas foram publicadas nos

seguintes artigos: Edson Floriano *et al.*: Privacidade em dados armazenados em memória compartilhada através de espaços de tupla, *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2017 [35] e Edson Floriano *et al.*: Providing privacy on the tuple space model, *Journal of Internet Services and Applications*, 8(19):1–16, 2017 [36].

6.2 Perspectivas Futuras

De modo geral, acreditamos que este trabalho pode ter uma ampla aplicação na área de sistemas distribuídos baseados em coordenação devido ao nível de segurança e funcionalidades alcançados, bem como aos seus ganhos expressivos de desempenho. Vislumbramos ainda como trabalhos futuros:

- A investigação de outros esquemas criptográficos que possam melhorar ainda mais as soluções propostas. Como exemplo, esquemas que possam combinar funcionalidades ou apresentar desempenho melhores;
- Projetar e implementar outros mecanismos de CDE utilizando estas soluções, como protocolos de barreiras distribuídas e eleição de líderes, tornando a coordenação distribuída ainda mais segura;
- Utilizar o DepSpace com as soluções propostas em sistemas práticos, como na implementação distribuída de controladores no ambiente de Redes Definidas por Software (*Software Defined Networks* - SDN). Neste ambiente, a preocupação com segurança e tolerância a falhas pode ser expressa na forma de garantir que a comunicação entre o plano de controle e o de dados ocorra de modo efetivo, mesmo na presença de adversários que possam tentar tomar conhecimento das regras instaladas ou alterá-las para obter algum benefício.

Referências

[1] Agrawal, Rakesh, Jerry Kiernan, Ramakrishnan Srikant e Yirong Xu: *Order preserving encryption for numeric data.* Em *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, páginas 563–574, New York, NY, USA, 2004. ACM, ISBN 1-58113-859-8. `http://doi.acm.org/10.1145/1007568.1007632`. 14

[2] Alchieri, E. A. P., A. N. Bessani e J. d. S. Fraga: *A dependable infrastructure for cooperative web services coordination.* Em *IEEE International Conference on Web Services*, páginas 21–28, 2008. 2

[3] Alves, Pedro Geraldo Morelli Rodrigues e Diego F. Aranha: *A framework for searching encrypted databases.* Em *XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSEG 2016)*, páginas 142–155, 2016. 12, 15, 16, 17, 32, 34, 36

[4] Analytics, N1: *A Java library for Paillier partially homomorphic encryption.* GitHub, 2017. `https://github.com/n1analytics/javallier`. 34, 51

[5] Avizienis, Algirdas, Jean Claude Laprie, Brian Randell e Carl Landwehr: *Basic concepts and taxonomy of dependable and secure computing.* IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, março 2004. xi, 1, 5, 6, 23

[6] Bakken, David E. e Richard D. Schlichting: *Supporting Fault-Tolerant Parallel Programing in Linda.* IEEE Transactions on Parallel and Distributed Systems, 6(3):287–302, março 1995. 22, 43

[7] Barker, Elaine B.: *Sp 800-57. recommendation for key management, part 1: General (revision 4).* Relatório Técnico, Gaithersburg, MD, United States, 2016. 29

[8] Barker, Elaine B. e Allen L. Roginsky: *Nist sp 800-131a revision 1. transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths.* Relatório Técnico, Gaithersburg, MD, United States, 2015. 36

[9] Bellare, Mihir, Ran Canetti e Hugo Krawczyk: *Keying hash functions for message authentication.* Em Koblitz, Neal (editor): *16th Annual International Cryptology Conference (CRYPTO 1996)*, volume 1109 de *LNCS*, páginas 1–15. Springer, 1996. 6

[10] Bellare, Mihir, Anand Desai, David Pointcheval e Phillip Rogaway: *Relations among notions of security for public-key encryption schemes.* Em *Advances in Cryptology*

- *CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, páginas 26–45, 1998. http://dx.doi.org/10.1007/BFb0055718. 11

[11] Bellare, Mihir, Tadayoshi Kohno e Chanathip Namprempre: *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, maio 2004, ISSN 1094-9224. http://doi.acm.org/10.1145/996943.996945. 12

[12] Bessani, Alysson, Jo
btxfnamespacelong ao Sousa e Eduardo E. P. Alchieri: *State machine replication for the masses with bft-smart.* Em *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '14, páginas 355–362, Washington, DC, USA, 2014. IEEE Computer Society, ISBN 978-1-4799-2233-8. http://dx.doi.org/10.1109/DSN.2014.43. 25

[13] Bessani, Alysson Neves, Eduardo Pelinson Alchieri, Miguel Correia e Joni da Silva Fraga: *DEPSPACE: A byzantine fault-tolerant coordination service.* European Conference on Computer Systems - EuroSys, páginas 163–176, 2008. x, 1, 2, 3, 22, 23, 24, 29, 39, 41, 42, 43, 55

[14] Bessani, Alysson Neves, Miguel Correia, Joni Silva Fraga e Lau Cheuk Lung: *Sharing memory between Byzantine processes using policy-enforced tuple spaces.* Em *Proceedings of 26th IEEE International Conference on Distributed Computing Systems - ICDCS 2006*, julho 2006. 2, 24

[15] Bettini, Lorenzo: *Data Privacy in Tuple Space Based Mobile Agent Systems.* Em R.Focardi e G.Zavattaro (editores): *2nd Int. Workshop on Security Issues in Coordination Models, Languages, and Systems (SecCo)*, número 568 em *ENTCS*. Elsevier, 2004. http://rap.dsi.unifi.it/~bettini/bibliography/files/cryptoagents.pdf. 22

[16] BFT-Smart: *DepSpace (Dependable Tuple Space).* GitHub, 2015. https://github.com/bft-smart/depspace. 36

[17] Boldyreva, Alexandra, Nathan Chenette, Younho Lee e Adam O'Neill: *Order-preserving symmetric encryption.* Cryptology ePrint Archive, Report 2012/624, 2012. http://eprint.iacr.org/2012/624. 14, 15, 34

[18] Boldyreva, Alexandra, Nathan Chenette e Adam O'Neill: *Order-preserving encryption revisited: Improved security analysis and alternative solutions.* Em *Proceedings of the 31st Annual Conference on Advances in Cryptology*, CRYPTO'11, páginas 578–595, Berlin, Heidelberg, 2011. Springer-Verlag, ISBN 978-3-642-22791-2. http://dl.acm.org/citation.cfm?id=2033036.2033080. 14, 15

[19] Boneh, Dan, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry e Joe Zimmerman: *Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation.* Cryptology ePrint Archive, Report 2014/834, 2014. http://eprint.iacr.org/2014/834. 14, 15

[20] Boneh, Dan e Victor Shoup: *A graduate course in applied cryptography.* `https://crypto.stanford.edu/~dabo/cryptobook/draft_0_2.pdf`, 2015. Acessado: 2016-11-30. 6, 8, 9, 11, 29, 31

[21] Burrows, Mike: *The chubby lock service for loosely-coupled distributed systems.* Em *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, páginas 335–350, Berkeley, CA, USA, 2006. USENIX Association, ISBN 1-931971-47-1. `http://dl.acm.org/citation.cfm?id=1298455.1298487`. 27

[22] Busi, Nadia, Roberto Gorrieri, Roberto Lucchi e Gianluigi Zavattaro: *SecSpaces: a Data-Driven Coordination Model for Environments Open to Untrusted Agents.* Em *Electronic Notes in Theoretical Computer Science*, volume 68, 2003. 2, 22, 43

[23] Castro, Miguel e Barbara Liskov: *Practical Byzantine fault-tolerance and proactive recovery.* ACM Transactions Computer Systems, 20(4):398–461, novembro 2002. 43

[24] Chenette, Nathan, Kevin Lewi, Stephen A. Weis e David J. Wu: *Practical order-revealing encryption with limited leakage.* Cryptology ePrint Archive, Report 2015/1125, 2015. `http://eprint.iacr.org/2015/1125`. 14, 15

[25] Coulouris, George, Jean Dollimore, Tim Kindberg e Gordon Blair: *Sistemas Distribuídos - Conceitos e Projeto.* Bookman, 2013. 18, 20

[26] Cristian, Flavin: *Understanding fault-tolerant distributed systems.* Commun. ACM, 34(2):56–78, fevereiro 1991, ISSN 0001-0782. `http://doi.acm.org/10.1145/102792.102801`. 19

[27] Darrel Hankerson, Alfred J. Menezes e Scott Vanstone: *Guide to Eliptic Curve Cryptography.* Springer-Verlag, 2004. 6, 7

[28] De Nicola, Rocco, Gian Luigi Ferrari e Rosario Pugliese: *KLAIM: A Kernel Language for Agents Interaction and Mobility.* IEEE Transactions on Software Engineering, 24(5):315–330, maio 1998. 2, 22, 43

[29] Diffie, W. e M. E. Hellman: *New Directions in Cryptography.* IEEE Transactions on Information Theory, IT-22:644–654, 1976. 7

[30] Distler, Tobias, Christopher Bahn, Alysson Bessani, Frank Fischer e Flavio Junqueira: *Extensible distributed coordination.* Em *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, 2015. x, 2, 22, 23, 29, 45, 46, 47, 50, 54

[31] ElGamal, Taher: *A public key cryptosystem and a signature scheme based on discrete logarithms.* IEEE Transactions on Information Theory, 31(4):469–472, 1985. 16, 32

[32] FIPS_PUB_196: *Advanced Encryption Standard.* Federal Information Processing (FIPS) Publication 196, 1997. 6

[33] FIPS_PUB_81: *Data Encryption Standard (DES).* Federal Information Processing (FIPS) Publication 81, 1980. 6, 25, 29

[34] Floriano, Edson: *blkOreJNI - A Java interface to the C implementation of Lewi and Wu FastORE through JNI.* GitHub, 2017. `https://github.com/florianoefsj/blkOreJNI`. 34, 51

[35] Floriano, Edson, Eduardo Alchieri, Diego Aranha e Priscila Solis: *Privacidade em dados armazenados em memória compartilhada através de espaços de tupla.* Em *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2017. 56

[36] Floriano, Edson, Eduardo Alchieri, Diego Aranha e Priscila Solis: *Providing privacy on the tuple space model.* Journal of Internet Services and Applications, 8(19):1–16, 2017. 37, 56

[37] Freeman, Eric, Ken Arnold e Susanne Hupfer: *JavaSpaces Principles, Patterns, and Practice.* Addison-Wesley Longman Ltd., Essex, UK, UK, 1st edição, 1999, ISBN 0201309556. 22

[38] G1: *Yahoo anuncia vazamento de dados que atinge 500 milhões de usuários*, 2016. `http://g1.globo.com/tecnologia/noticia/2016/09/yahoo-anuncia-vazamento-de-dados-que-atinge-500-milhoes-de-usuarios.html`, acesso em 2017-01-18. 1, 5

[39] Gelernter, David: *Generative Communication in Linda.* ACM Transactions on Programing Languages and Systems, 7(1):80–112, janeiro 1985. 1, 21, 22

[40] Ghemawat, Sanjay, Howard Gobioff e Shun Tak Leung: *The google file system.* SIGOPS Oper. Syst. Rev., 37(5):29–43, outubro 2003, ISSN 0163-5980. `http://doi.acm.org/10.1145/1165389.945450`. 27

[41] GigaSpaces: *GigaSpaces Homepage.* Disponível em `http://www.gigaspaces.com/`, Dez 2016. 22, 43

[42] Herlihy, Maurice: *Wait-free synchronization.* ACM Trans. Program. Lang. Syst., 13(1):124–149, janeiro 1991, ISSN 0164-0925. `http://doi.acm.org/10.1145/114005.102808`. 46

[43] Herlihy, Maurice, Victor Luchangco e Mark Moir: *Obstruction-free synchronization: Double-ended queues as an example.* Em *23rd International Conference on Distributed Computing Systems (ICDCS 2003), 19-22 May 2003, Providence, RI, USA*, páginas 522–529, 2003. `https://doi.org/10.1109/ICDCS.2003.1203503`. 46

[44] Hunt, Patrick, Mahadev Konar, Flavio P. Junqueira e Benjamin Reed: *Zookeeper: Wait-free coordination for internet-scale systems.* Em *2010 USENIX Annual Technical Conference (ATC '10)*, página 145–158, 2010. 27

[45] *Information technology – security techniques – code of practice for information security management.* Standard, International Organization for Standardization, Geneva, CH, junho 2005. 5

[46] *Information technology – security techniques – code of practice for information security management.* Standard, International Organization for Standardization, Geneva, CH, junho 2005. 5

[47] *Information technology – security techniques – code of practice for information security controls.* Standard, International Organization for Standardization, Geneva, CH, outubro 2013. 5

[48] JavaSpaces: *JavaSpaces guide.* Disponível em `http://www.oracle.com/technetwork/articles/java/javaspaces-140665.html`, Dez 2016. 22, 43

[49] Lamport, Leslie: *Time, clocks, and the ordering of events in a distributed system.* Commun. ACM, 21(7):558–565, julho 1978, ISSN 0001-0782. `http://doi.acm.org/10.1145/359545.359563`. 18, 19, 20

[50] Lamport, Leslie, Robert Shostak e Marshall Pease: *The byzantine generals problem.* ACM Trans. Program. Lang. Syst., 4(3):382–401, julho 1982, ISSN 0164-0925. `http://doi.acm.org/10.1145/357172.357176`. 19

[51] Lewi, Kevin e David J. Wu: *FastORE - An Implementation of Order-Revealing Encryption.* GitHub, 2016. `https://github.com/kevinlewi/fastore`. 34, 51

[52] Lewi, Kevin e David J. Wu: *Order-revealing encryption: New constructions, applications, and lower bounds.* Em *ACM Conference on Computer and Communications Security (ACM CCS)*, 2016. 15, 17, 32, 34, 35, 51

[53] M. R. Alves, Pedro G. e Diego F. Aranha: *A framework for searching encrypted databases.* Journal of Internet Services and Applications, 9(1):1, Jan 2018, ISSN 1869-0238. `https://doi.org/10.1186/s13174-017-0073-0`. 15, 16, 17

[54] Macedo, R., J. Paulo, R. Pontes, B. Portela, T. Oliveira, M. Matos e R. Oliveira: *A practical framework for privacy-preserving nosql databases.* Em *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, páginas 11–20, Sept 2017. 16, 17

[55] Menezes, Alfred J., Scott A. Vanstone e Paul C. Van Oorschot: *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1ª edição, 1996, ISBN 0849385237. xi, 5, 6, 9

[56] Morais, Eduardo e Ricardo Dahab: *Encriptação homomórfica.* Em *Minicursos do XII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSEG 2012)*, páginas 151–195, 2012. 14, 15

[57] Naehrig, Michael, Kristin Lauter e Vinod Vaikuntanathan: *Can homomorphic encryption be practical?* Em *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, páginas 113–124, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-1004-8. `http://doi.acm.org/10.1145/2046660.2046682`. 14, 15, 16

[58] Naveed, Muhammad, Seny Kamara e Charles V. Wright: *Inference attacks on property-preserving encrypted databases.* Em *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, páginas 644–655, 2015. `http://doi.acm.org/10.1145/2810103.2813651`. 1, 13

[59] Paillier, Pascal: *Public-key cryptosystems based on composite degree residuosity classes.* Em *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, páginas 223–238, Berlin, Heidelberg, 1999. Springer-Verlag, ISBN 3-540-65889-0. `http://dl.acm.org/citation.cfm?id=1756123.1756146`. 16, 32, 34, 51

[60] Popa, Raluca Ada, Catherine M. S. Redfield, Nickolai Zeldovich e Hari Balakrishnan: *CryptDB: Protecting confidentiality with encrypted query processing.* Em *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, páginas 85–100, 2011. 12, 16

[61] Rivest, R. L., A. Shamir e L. Adleman: *A method for obtaining digital signatures and public-key cryptosystems.* Communications of the ACM, 21(2):120–126, 1978. 7, 14, 15

[62] Savvides, Savvas: *Order-preserving encryption in java.* GitHub. `https://github.com/ssavvides/jope`. 34

[63] Schneider, Fred B.: *Byzantine generals in action: Implementing fail-stop processors.* ACM Trans. Comput. Syst., 2(2):145–154, maio 1984, ISSN 0734-2071. `http://doi.acm.org/10.1145/190.357399`. 19

[64] Schneider, Fred B.: *Implementing fault-tolerant service using the state machine aproach: A tutorial.* ACM Computing Surveys, 22(4):299–319, dezembro 1990. 24, 43

[65] Schoenmakers, Berry: *A simple publicly verifiable secret sharing scheme and its application to electronic voting.* Em *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology - CRYPTO'99*, páginas 148–164, agosto 1999. 25, 29

[66] Segall, Edward J.: *Resilient distributed objects: Basic results and applications to shared spaces.* Em *Proceedings of the 7th Symposium on Parallel and Distributed Processing*, 1995. 22

[67] Silva, E. A. e M. Correia: *Leveraging an homomorphic encryption library to implement a coordination service.* Em *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, páginas 39–42, Oct 2016. 23

[68] Stevens, Marc, Elie Bursztein, Pierre Karpman, Ange Albertini e Yarik Markov: *The first collision for full sha-1.* Cryptology ePrint Archive, Report 2017/190, 2017. `https://eprint.iacr.org/2017/190`. 28

[69] Stinson, Douglas: *Cryptography: Theory and Practice,Third Edition.* CRC/C&H, 3ª edição, 2006. 7, 8, 11

[70] T. J. Lehman et al.: *Hitting the distributed computing sweet spot with TSpaces.* Computer Networks, 35(4):457–472, março 2001. 22, 43

[71] Tanenbaum, Andrew e Maarten van Steen: *Sistemas Distribuídos: princípios e paradigmas.* Pearson Prentice Hall. Pearson Education International, 2ª edição, 2007. 18, 19, 20

[72] Veríssimo, Paulo: *Dialogue on cyber policies between brazil and the eu: prospecting threats and opportunities of the cyberspace.* Dialogue on Cyber Policies, 2016. 1

[73] Vitek, Jan, Ciarán Bryce e Manuel Oriol: *Coordination processes with Secure Spaces.* Science of Computer Programming, 46(1-2):163–193, janeiro 2003. 2, 22, 43

[74] Washington, Lawence C.: *Elliptic curves: number theory and cryptography.* 2ª edição, 2008. 7

[75] White, Brian, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb e Abhijeet Joglekar: *An Integrated Experimental Environment for Distributed Systems and Networks.* Em *Proc. of 5th Symp. on Operating Systems Design and Implementations.* ACM, 2002. 36, 50

[76] Xu, Andrew e Barbara Liskov: *A design for a fault-tolerant, distributed implementation of Linda.* Em *Proc. of the 19th Symposium on Fault-Tolerant Computing*, páginas 199–206, junho 1989. 22, 43