Provendo Confidencialidade em Espaços de Tuplas Tolerantes a Intrusões*

Alysson Neves Bessani[1], **Eduardo Adílio Pelison Alchieri**[1], **Miguel Correia**[2], **Joni da Silva Fraga**[1], **Lau Cheuk Lung**[3]

[1] DAS, PGEEL, Universidade Federal de Santa Catarina - Florianópolis - SC - Brasil

[2]LASIGE, Faculdade de Ciências da Universidade de Lisboa - Lisboa - Portugal

[3]PPGIA, Pontifícia Universidade Católica do Paraná - Curitiba - PR - Brasil

{neves,alchieri,fraga}@das.ufsc.br, mpc@di.fc.ul.pt, lau@ppgia.pucpr.br

Resumo. *A coordenação por espaços de tuplas é um dos mais interessantes modelos de comunicação para sistemas distribuídos abertos, devido as suas características de desacoplamento espacial e temporal e ao seu poder de sincronização. Muitos destes sistemas estão sujeitos a faltas, ataques e intrusões, porém é fundamental que a estrutura de comunicação neles empregada permaneça provendo seu serviço corretamente mesmo na presença desses eventos. Para fornecer este nível de qualidade de serviço uma abordagem interessante é a tolerância a intrusões, onde o sistema é implementado por um conjunto de réplicas que provêm o serviço corretamente mesmo que uma parte delas sejam controladas por um adversário. Este trabalho apresenta um esquema de confidencialidade para espaços de tuplas tolerantes a intrusões baseado em compartilhamento de segredo, onde uma tupla (unidade de dados armazenada no espaço) não é revelada a partes não autorizadas mesmo que algumas das réplicas do espaço sejam faltosas. Visando validar este esquema, alguns experimentos que medem o impacto da inclusão do esquema em um espaço de tuplas tolerante a intrusões são apresentados.*

1. Introdução

A maioria dos sistemas distribuídos modernos têm características de sistemas abertos. Estes sistemas tipicamente são compostos por um número desconhecido de processos, executando em ambientes heterogêneos e não confiáveis, conectados através de redes também heterogêneas e não confiáveis, como a Internet. Embora muitas aplicações distribuídas continuam sendo desenvolvidas sobre simples primitivas de comunicação, como *sockets* TCP/IP e chamadas a procedimentos remotos, existe uma grande e importante demanda por ferramentas mais elaboradas, que permitam a construção de aplicações mais complexas de forma eficiente e rápida. Considerando a necessidade de tolerar desconexões, recuperar paradas de servidores e segurança contra ações maliciosas, a concepção de tais ferramentas torna-se uma tarefa ainda mais difícil.

Para suprir estes requisitos, é necessário uma abordagem onde as interações sejam desacopladas. Deste modo, são necessários modelos alternativos de coordenação. Dentre

*Trabalho realizado com o suporte do CNPq (processo 550114/2005-0).

estes modelos, a coordenação generativa [Gelernter, 1985] destaca-se devido a sua flexibilidade e simplicidade. Neste modelo, os processos interagem através de um espaço de memória compartilhado, chamado espaço de tuplas, onde estruturas genéricas de dados, chamadas tuplas, são armazenadas e recuperadas. A coordenação é desacoplada no tempo (os participantes não precisam estar ativos no mesmo instante) e no espaço (os participantes não precisam se conhecer).

Desde a introdução do modelo de coordenação através de espaços de tuplas, têm havido pesquisas sobre a introdução de tolerância a faltas neste modelo, tanto através da construção de espaços de tuplas tolerantes a faltas (usando replicação) quanto em mecanismos que permitam a construção de aplicações tolerantes a faltas sobre o espaço de tuplas (suportando transações). Mais recentemente, alguns esforços sobre espaços de tuplas seguros tem sido relatados na literatura [Busi et al., 2003, De Nicola et al., 1998, Vitek et al., 2003]. O objetivo destes trabalhos é garantir que processos não executem operações no espaço de tuplas sem permissão, usando mecanismos de controle de acesso em nível de espaço e de tupla.

Estes trabalhos em tolerância a faltas e segurança para espaço de tuplas têm um foco limitado em pelo menos dois sentidos: eles consideram apenas faltas acidentais por parada e ataques simples (acesso inválido); eles tratam de tolerância a faltas **ou** de segurança. Uma abordagem mais abrangente consiste em agrupar mecanismos de tolerância a faltas (como replicação) e segurança (como criptografia) e implementar sistemas que permanecem provendo serviços corretamente mesmo que uma parte de seus componentes sejam atacados, invadidos e controlados por adversários. Esta abordagem é conhecida como **tolerância a intrusões** [Fraga and Powell, 1985, Veríssimo et al., 2003].

Neste sentido, recentemente foi introduzido um espaços de tuplas tolerante a faltas maliciosas, i.e. intrusões [Bessani et al., 2006b]. Estes espaços são implementados por um conjunto de n servidores e permanecem operacionais enquanto menos de 1/3 destes servidores sejam faltosos (o número máximo de servidores que podem falhar é comumente denotado por f). Estes trabalhos tratam basicamente da manutenção da integridade e da disponibilidade do espaço de tuplas enquanto serviço. Neste trabalho consideramos o problema da confidencialidade e apresentamos um esquema de confidencialidade para espaços de tuplas tolerantes a intrusões, baseado em compartilhamento de segredo. Este esquema garante que os dados armazenados no espaço de tuplas não sejam revelados a partes não autorizadas mesmo que algumas réplicas do espaço sejam faltosas. Além disso, são discutidas formas para integração deste esquema nos espaços de tuplas tolerantes a faltas maliciosas previamente desenvolvidos. A validação do esquema é feita através da implementação de um protótipo e a realização de alguns experimentos para medição do impacto do mecanismo de confidencialidade na latência de um espaço de tuplas tolerante a intrusões.

O texto esta organizado da seguinte forma: Na seção 2 temos a definição do problema, onde os atributos necessários para um espaço de tuplas ter segurança de funcionamento são discutidos. Esta seção ainda aborda os mecanismos utilizados na concretização de tal espaço de tuplas e algumas dificuldades na provisão de confidencialidade. A seção 3 apresenta o esquema de confidencialidade proposto e a sua aplicação em duas técnicas de replicação utilizadas em espaços de tuplas tolerantes a intrusões. A seção 4 descreve a implementação dos protocolos. A seção 5 apresenta medidas de desempenho do esquema

integrado a um espaço de tuplas tolerante a intrusões. Alguns trabalhos relacionados são relatados na seção 6. Finalmente, a seção 7 apresenta as conclusões do trabalho.

2. Definição do Problema

Coordenação é um paradigma clássico de sistemas distribuídos, baseado na idéia de separar as atividades do sistema em dois tipos: computação e coordenação [Gelernter and Carriero, 1992]. O modelo de comunicação generativa [Gelernter, 1985] (também chamada coordenação generativa), originalmente introduzido no contexto da linguagem de programação LINDA [Gelernter, 1985], utiliza um espaço de memória compartilhado, chamado espaço de tuplas, onde estruturas genéricas de dados, chamadas tuplas, são armazenadas e recuperadas. Esta coordenação é desacoplada no tempo (os participantes não precisam estar ativos no mesmo instante) e no espaço (os participantes não precisam se conhecer).

Uma tupla $t = \langle f_1, f_2, ..., f_n \rangle$ é um conjunto de campos ordenados. Cada campo f_i da tupla pode ser um atual, um formal ou o símbolo especial '$*$'. Um atual contém um valor. Um formal representa uma variável onde será colocado o valor de uma tupla obtida do espaço. Um campo deste tipo é denotado pelo nome da variável precedido de um '?'. O símbolo especial '$*$' representa qualquer valor. Uma tupla em que todos os parâmetros são atuais é chamada de entrada (*entry*). Um molde (*template*) é uma tupla que contém um ou mais campos formais ou '$*$', sendo representada por $\bar{t}$. Uma tupla t e um molde $\bar{t}$ combinam somente se eles tiverem o mesmo número de campos e todos os campos com valores definidos em $\bar{t}$ são iguais aos valores dos campos correspondentes em t. Esta combinação é denotada por $m(t, \bar{t})$.

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

2.1. Segurança de Funcionamento em Espaço de Tuplas

Um espaço de tuplas é dito com segurança de funcionamento se ele provê os **atributos de segurança de funcionamento** [Avizienis et al., 2004]. Como muitos outros sistemas, alguns destes atributos não são aplicáveis ou são ortogonais ao projeto do espaço de tuplas (por exemplo, *safety* e manutenibilidade). Os atributos relevantes neste caso são:

- **Confiabilidade:** as operações realizadas no espaço de tuplas fazem com que seu estado se modifique de acordo com sua especificação;
- **Disponibilidade:** o espaço de tuplas sempre está pronto para executar as operações requisitadas por partes autorizadas;

- **Integridade:** nenhuma alteração imprópria no estado de um espaço de tuplas pode ocorrer, i.e. o estado de um espaço de tuplas só pode ser alterado através da execução das operações suportadas por ele;
- **Confidencialidade:** o conteúdo de (alguns) campos de uma tupla não podem ser revelados a partes não autorizadas.

A dificuldade em garantir estes atributos advém da ocorrência de faltas, de natureza acidental (um *bug* no software ou uma parada no servidor) ou maliciosa (um invasor que modifica uma tupla no servidor). O objetivo é evitar que estas faltas causem uma falha no espaço de tuplas, i.e. que um ou mais destes atributos sejam violados. Deste modo, é garantido que o espaço de tuplas se comportará de acordo com sua especificação, mesmo na presença de alguns servidores faltosos (no máximo f) e de um número ilimitado de clientes faltosos (a arquitetura geral deste modelo é apresentada na figura 1). Faltas maliciosas são particularmente difíceis de serem tratadas já que não é possível definir premissas a respeito do que um invasor pode ou não fazer no sistema [Veríssimo et al., 2003]. Estas faltas são usualmente modeladas como a classe mais genérica de faltas – faltas arbitrárias ou Bizantinas [Lamport et al., 1982] – de tal forma que a solução aqui proposta é genérica em termos do tipo de falta tolerada.

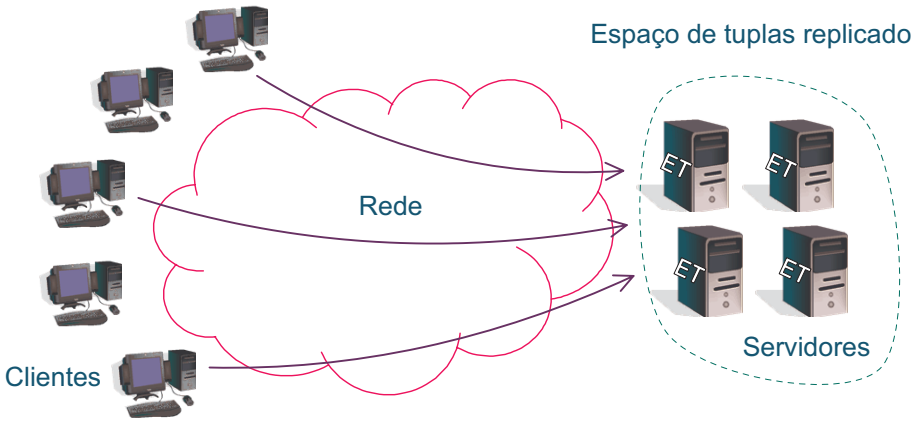


Figura 1. Espaço de tuplas com segurança de funcionamento.

O significado dos atributos de confiabilidade e disponibilidade são claros e podem ser facilmente entendidos, entretanto integridade e confidencialidade requerem alguma discussão. Uma alteração no espaço de tuplas é dita apropriada (vs. inapropriada) se e somente se ela é o resultado de uma das operações definidas para o espaço de tuplas (escrita, leitura ou remoção). Por construção, apenas alterações apropriadas são permitidas no espaço de tuplas, mesmo no caso de um servidor faltoso permitir alterações inapropriadas na sua réplica local, pois os servidores corretos não permitirão que esta alteração tenha efeito [Castro and Liskov, 2002, Bessani et al., 2006b]. Sendo assim, o atributo de integridade é garantido.

Confidencialidade é um dos atributos mais difícil de ser suportado, pois requer privacidade da informação, mesmo que esta esteja replicada. No entanto, existem técnicas de criptografia que permitem este sigilo da informação, mesmo na presença de servidores faltosos. Nas seções seguintes, discutiremos como a confidencialidade pode ser provida.

2.1.1. Mecanismos para Segurança de Funcionamento

A segurança de funcionamento de um espaço de tuplas pode ser implementada utilizando uma combinação de diversos mecanismos. O mecanismo mais básico usado para implementar um espaço de tuplas com segurança de funcionamento é a **replicação**: o espaço de tuplas é mantido replicado em um conjunto de n servidores de tal forma que a falha em alguns deles não viole a confiabilidade, disponibilidade e integridade do sistema. A idéia é utilizar um esquema de replicação que permita que mesmo em caso de falha em alguns servidores (sendo controlados por invasores e agindo maliciosamente ou apenas parando), o espaço de tuplas permanece disponível com suas operações seguindo sua especificação (mantendo confiabilidade e integridade). Obviamente, existe um limite máximo f de réplicas que podem falhar, este limite depende da solução de replicação empregada.

Outro mecanismo fundamental para segurança de funcionamento, especialmente para manutenção da integridade e confidencialidade, é o **controle de acesso**. Este tipo de mecanismo é necessário para prevenir clientes não autorizados de inserir (out), remover (in e inp) ou ler (rd, rdp) do espaço de tuplas. Controle de acesso é usualmente implementado em servidores replicados através de um monitor de referência, que implementa a mesma política, instalado em cada um dos servidores.

O terceiro tipo de mecanismo é a **criptografia**, que é usada para garantir confiabilidade nas comunicações e a confidencialidade das tuplas. Implementar confidencialidade em um espaço de tuplas replicado não é simples por diversas razões. A primeira, e mais importante, é que não podemos confiar nos servidores individualmente para garantir a confidencialidade das tuplas, uma vez que até f servidores podem falhar, possivelmente revelando as tuplas armazenadas a partes não autorizadas. A segunda complicação advém da necessidade de combinação entre tuplas e moldes. Campos encriptados geralmente não podem ser comparados com campos de um molde. Além disso, algumas políticas de acesso de granularidade fina [Bessani et al., 2006a] podem impor limites nos campos que podem ser encriptados.

2.1.2. Confidencialidade em Espaço de Tuplas Tolerante a Intrusões

A utilização de replicação para manter a disponibilidade e a integridade de um sistema é muitas vezes vista como um impedimento para a implementação da confidencialidade do mesmo. A razão para isto é simples: se uma informação secreta não é armazenada em um, mas em vários servidores, é possivelmente mais fácil para um atacante obtê-la (não mais difícil). Assim, a implementação de confidencialidade em um espaço de tuplas replicado não é trivial. Várias soluções aparentemente simples não funcionam neste cenário ou são simplesmente inaceitáveis quando pensamos no modelo de coordenação baseado em espaço de tuplas. Duas das mais intuitivas destas soluções seriam:

- **Fazer com que cada cliente cifrasse os campos das tuplas inseridos por ele usando uma chave secreta compartilhada ou sua chave privada**: o problema desta solução está no fato de que ela requer que todos os clientes que queiram ler e/ou remover uma tupla cifrada conheçam a chave secreta ou as chaves pública (para que campos cifrados possam ser decifrados) e privada (para encriptar campos de moldes e as regras de combinação possam ser usadas). Além do problema

prático da dificuldade em se realizar a distribuição destas chaves, esta solução acaba com a propriedade de anonimato do modelo de coordenação baseado em espaço de tuplas, que define que os clientes não precisam conhecer uns aos outros para interagirem;

- **Cifrar a comunicação cliente-servidores e deixar que estes últimos cifrem os campos das tuplas com suas próprias chaves**: Esta solução não é aceitável em nosso modelo de sistema porque queremos manter a confidencialidade mesmo que alguns servidores sejam controlados pelo adversário, e estes servidores poderiam decifrar e revelar o conteúdo de campos confidenciais das tuplas.

Note que os requisitos inerentes a um espaço de tuplas tolerante a intrusões, especialmente a possibilidade de alguns servidores estarem completamente sobre controle de um adversário malicioso, torna todos os mecanismos de segurança de um espaço de tuplas propostos previamente [Vitek et al., 2003, Busi et al., 2003] inúteis, uma vez que todos eles assumem servidores confiáveis.

3. Esquema de Confidencialidade

Nas seções anteriores, apresentamos os requisitos necessários para o funcionamento seguro de um espaço de tuplas. Nesta seção, trataremos especificamente da provisão do atributo de confidencialidade em espaços de tuplas replicados visando tolerância a intrusões. Iniciaremos nossa discussão apresentando as premissas gerais do sistema e as ferramentas criptográficas utilizadas no esquema, para então apresentar uma visão geral de seu funcionamento. A seguir, os algoritmos de escrita e leitura de tuplas com confidencialidade são apresentados. A seção termina com a apresentação de como integrar o esquema de confidencialidade a espaços de tuplas tolerantes a intrusões baseados em diferentes modelos de replicação.

3.1. Modelo de Sistema

Consideramos um espaço de tuplas implementado por n servidores $U = \{s_1, ..., s_n\}$ e um conjunto ilimitado de clientes $\Pi = \{p_1, p_2, ...\}$. Todos os processos se comunicam através de canais confiáveis e autenticados[1]. O esquema de confidencialidade não requer nenhum tipo de premissa temporal[2].

Em termos de falhas, assumimos um sistema sujeito a faltas Bizantinas com autenticação [Lamport et al., 1982]: um adversário malicioso que pode corromper e controlar até $f \leq \lfloor \frac{n-1}{3} \rfloor$ servidores e um número ilimitado de clientes.

Finalmente, assume-se que existe um mecanismo de controle de acesso instalado nos servidores que impede que processos maliciosos leiam tuplas que eles não têm direito de acesso. Note que os servidores faltosos podem não respeitar esse mecanismo de controle de acesso e revelar os dados neles armazenados a qualquer processo.

3.2. Ferramentas Criptográficas

Esta seção detalha as principais ferramentas criptográficas utilizadas no esquema de confidencialidade desenvolvido.

[1] Que podem ser facilmente implementado através de SSL ou TCP/IP com IPSec.

[2] Muito embora um espaço de tuplas replicado só possa ser implementado de forma deterministica se assumir algum nível de garantia temporal [Bessani et al., 2006b].

3.2.1. Resumo Criptográfico

Uma **função criptográfica de resumo resistente a colisões** H é uma função que mapeia uma entrada de tamanho arbitrário em uma saída de tamanho fixo. Esta função deve satisfazer duas propriedades básicas:

1. **Resistência a colisão**: é computacionalmente inviável[3] obter dois valores $v \neq v'$ de tal forma que $\mathsf{H}(v) = \mathsf{H}(v')$;
2. **Unidirecionalidade**: dada uma saída da aplicação da função de resumo, é computacionalmente inviável encontrar a entrada que produz essa saída.

Durante o texto, chamaremos o resultado da aplicação da função de resumo em um valor de resumo criptográfico (ou resumo) do valor. Um exemplo de uma função de resumo com essas características é a função SHA-1, que gera uma saída de 160 bits.

3.2.2. Compartilhamento de Segredo Verificável Publicamente

O outro esquema criptográfico utilizado neste trabalho é o **compartilhamento de segredo verificável publicamente** (*publicly verifiable secret sharing scheme* - PVSS) [Schoenmakers, 1999]. Este esquema é utilizado para garantir a confidencialidade das tuplas armazenadas no espaço replicado mesmo em face a existência de servidores maliciosos. Em um esquema (n, k)-PVSS, um **distribuidor** (*dealer*) distribui fragmentos (*shares*) de um segredo para n partes distintas, chamadas **portadores** (*share holders*), usando um protocolo de distribuição. Posteriormente, um **combinador** (*combiner*) obtém pelo menos k fragmentos para reconstruir o segredo. O esquema é dito "verficável publicamente" porque qualquer parte pode verificar se os fragmentos estão corrompidos (não apenas os n portadores). Uma propriedade fundamental deste tipo de esquema é que nenhuma informação sobre o segredo compartilhado é obtida com menos de k fragmentos. Este esquema é baseado nas seguintes primitivas (denotamos por x_i e y_i, respectivamente, as chaves privadas e públicas de cada portador i):

- $\mathsf{share}(y_1, ..., y_n, s)$ é usada para gerar os n fragmentos $\{s_1, ..., s_n\}$ do segredo s e a prova criptográfica $PROOF_s$ de que estes segredos são corretos;
- $\mathsf{verifyD}(s_i, PROOF_s)$ é usada pelo portador i para verificar se o fragmento s_i distribuído pelo distribuidor é correto;
- $\mathsf{prove}(s_i, x_i, PROOF_s)$ é usado para gerar uma prova criptográfica $PROOF_s^i$, a qual é usada para provar que o fragmento s_i, apresentado pelo portador i para o combinador, é correto;
- $\mathsf{verifyS}(s_i, y_i, PROOF_s, PROOF_s^i)$ é usada para verificar se um fragmento s_i obtido do portador i é realmente o fragmento distribuído pelo combinador juntamente com a prova $PROOF_s$;
- $\mathsf{combine}(s_1, ..., s_k)$ é usada para combinar um conjunto de k fragmentos e recuperar o segredo s.

Para a implementação do esquema de confidencialidade no espaço de tuplas replicado, utilizamos o protocolo proposto em [Schoenmakers, 1999]. A segurança deste protocolo se baseia nas premissas do problema Diffie-Helman [Diffie and Hellman, 1976] e ele é provado seguro no modelo dos oráculos aleatórios [Bellare and Rogaway, 1993].

[3]Não existe um algoritmo que possa executar esta tarefa de forma eficiente [Goldereich, 2001].

3.3. Visão Geral

A solução proposta neste artigo segue a idéia de deixar os servidores cuidarem da confidencialidade, entretanto, ao invés de confiar nos servidores individualmente, confiamos em um **conjunto** de servidores, seguindo o paradigma de **confiança distribuída** [Schneider and Zhou, 2005]. Usamos o protocolo de compartilhamento de segredo verificável publicamente apresentado na seção 3.2.2, considerando $k = f + 1$. Nele, cada servidor de espaço de tuplas s_i tém uma chave privada x_i e uma chave pública y_i. Os clientes conhecem as chaves públicas de todos os servidores e fazem o papel dos distribuidores do protocolo, obtendo um conjunto de *shares* dos campos (função share). Todo campo da tupla pode ser decifrado com $k = f+1$ *shares* (função combine), portanto uma coalizão de servidores maliciosos não pode revelar o conteúdo de um campo confidencial (assumimos no máximo f servidores faltosos).

Além de garantir a confidencialidade da tupla mesmo com uma parte dos servidores controlada pelo adversário, um esquema de confidencialidade para espaço de tuplas com segurança de funcionamento deve garantir o acesso por conteúdo as tuplas (através das regras de combinação), mesmo na existência de campos cifrados. Quando um cliente invoca a operação $out(t)$, ele escolhe um dos três tipos de proteção para cada um dos campos de t:

- **público**: o campo não é cifrado e portanto seu valor pode ser comparado arbitrariamente (p.ex. se ele pertence a um intervalo) e revelado a partes maliciosas[4];
- **comparável**: o campo é cifrado e um resumo criptográfico (*hash*) de seu valor é armazenado juntamente com a tupla de tal forma que comparações de igualdade sejam possíveis (detalhes a seguir);
- **privado**: o campo é cifrado e nenhuma informação derivada dele é armazenada, logo seu conteúdo não pode ser usado em comparações.

A idéia por trás dos campos marcados como comparáveis é a seguinte. Suponha que o cliente p_1 queira inserir uma tupla t no espaço com um único campo comparável f_1. Assumindo a existência de uma função de resumo resistente a colisão $\mathsf{H}(v)$, p_1 envia t cifrada juntamente com $\mathsf{H}(f_1)$ aos servidores. Suponha que mais tarde um cliente p_2 invoque $rd(\bar{t})$ e que o espaço de tuplas precise verificar se $\bar{t}$ combina com t. Assumindo que $\bar{t}$ tenha apenas um único campo $\overline{f}_1$, p_2 calcula $\mathsf{H}(\overline{f}_1)$ e envia este valor aos servidores do espaço de tupla que verificam se este resumo combina com $\mathsf{H}(f_1)$. Este esquema funciona para a regra clássica de combinação do modelo de espaço de tuplas (comparações de igualdade), porém claramente não funciona com outras comparações como desigualdades[5]. Além disso, este esquema tem outra limitação: apesar das funções de resumo serem unidirecionais, se o domínio dos valores que um campo pode conter são conhecidos e limitados, um ataque de força bruta pode revelar seu conteúdo. Por exemplo, suponha que um campo pode conter apenas valores de 32 bits. Um atacante pode simplesmente calcular os resumos de 2^{32} possíveis valores para descobrir qual o valor do campo armazenado. Este problema é a principal motivação para o não uso de campos tipados no espaço de

[4]Este tipo de campo pode ser necessário, por exemplo, para o emprego de políticas de acesso de granularidade fina ([Bessani et al., 2006a]).

[5]Para que funcionasse precisaríamos de uma função de resumo que mantivesse o resultado da comparação entre valores quando seus resumos fossem comparados. Por exemplo, para desigualdades, se $v > v'$ então $\mathsf{H}(v) > \mathsf{H}(v')$.

tuplas considerado neste artigo. Adicionalmente, a limitação dos campos comparáveis é razão pela qual definimos os campos privados: nenhum resumo é enviado aos servidores e portanto comparações são impossíveis, garantindo a confidencialidade destes campos.

3.4. Algoritmo Abstrato

A idéia principal do esquema de confidencialidade proposto neste artigo é fazer com que o processo que insere a tupla no espaço atue como um distribuidor no esquema PVSS, gerando n fragmentos da tupla e distribuindo-os entre os servidores. Um processo que deseja ler uma tupla, coleta $f + 1$ destes fragmentos e combina-os para obter a tupla. É necessário um esquema de verificação pública para permitir que um cliente verifique se o fragmento retornado pelo servidor é correto (nos esquemas de compartilhamento de segredos verificável "normais" apenas os portadores, os servidores neste caso, podem verificar se um fragmento está corrompido). Isto é muito importante uma vez que estamos considerando um sistema aberto com um número ilimitado de clientes.

Os algoritmos 1 e 2 descrevem informalmente o esquema de confidencialidade para a escrita e leitura de tuplas, respectivamente. O algoritmo para remoção é similar ao de leitura.

Algoritmo 1 Esquema de confidencialidade na inserção da tupla t

---

1. p gera n fragmentos ts_i e a prova de corretude $PROOF_t$, usando a função $\mathsf{share}(y_1, ..., y_n, t)$.
2. p computa o *fingerprint* de $t = \langle f_1, ..., f_m \rangle$, definido como:

   $t_h = \langle \mathsf{F}(f_1), ..., \mathsf{F}(f_m) \rangle$, onde: $\mathsf{F}(f_i) \leftarrow \begin{cases} * & \text{se } f_i = * \\ f_i & \text{se } f_i \text{ é público ou formal} \\ \mathsf{H}(f_i) & \text{se } f_i \text{ é comparável} \\ \mathsf{PR} & \text{se } f_i \text{ é privado} \end{cases}$
3. p envia uma mensagem $\langle ts_i, t_h, PROOF_t \rangle$, para cada servidor do sistema s_i.
4. Um servidor s_i aceita a tupla t inserida por p, se $\mathsf{verifyD}(ts_i, PROOF_t)$ retornar $true$.
5. Cada servidor s_i que aceitar a tupla t, calcula $PROOF_t^i = \mathsf{prove}(ts_i, x_i, PROOF_t)$ e armazena $\langle t_h, ts_i, PROOF_s, PROOF_t^i, p \rangle$.

Algoritmo 2 Esquema de confidencialidade na leitura da tupla t

---

1. p calcula o *fingerprint* $\bar{t}_h$ para o molde $\bar{t}$, conforme definido no passo 2 do Algoritmo 1.
2. p envia uma mensagem de leitura aos servidores, utilizando $\bar{t}_h$ como molde.
3. Cada servidor s_i retorna um conjunto contendo um elemento $e = \langle ts_i, t_h, PROOF_t, PROOF_t^i \rangle$ para cada tupla com *fingerprint* t_h, tal que $m(t_h, \bar{t}_h)$.
4. p recebe as respostas de um conjunto de servidores e escolhe uma tupla t que possui $f + 1$ fragmentos corretos, i.e., $f + 1$ servidores responderam o mesmo t_h e, para cada servidor s_i, sua prova satisfaz $\mathsf{verifyS}(ts_i, y_i, PROOF_t^i)$. O processo p combina estes $f + 1$ fragmentos corretos usando $\mathsf{combine}(ts_1, ..., ts_{f+1})$ e obtém a tupla t.
5. p verifica se t_h é o *fingerprint* de t. Se não for, p envia uma mensagem INVALID-TUPLE para todos os servidores, com todos os dados coletados durante a leitura, para provar que esta tupla está corrompida, e escolhe outra tupla para ser lida (ou retorna $\perp$ se não existir outra tupla). Esta tupla t é removida dos servidores caso for verificado que t_h não é o *fingerprint* de t. Adicionalmente, o processo que inseriu t é colocado em uma "*black list*" e suas mensagens futuras são ignoradas.

Existem vários pontos sobre o esquema de confidencialidade que merecem alguma discussão. Primeiro, a maior parte do processamento, como a geração e combinação dos fragmentos do segredo, é realizada no processo cliente. Deste modo, a utilização deste esquema não deve causar impactos severos na escalabilidade de um sistema.

O segundo ponto, também relacionado com a escalabilidade, é que não usamos nenhum protocolo com complexidade de troca de mensagens quadrática, os quais não apresentam uma boa escalabilidade. O esquema é baseado em protocolos que apresentam uma complexidade de mensagens $O(n)$ (o protocolo não requer que os servidores se comuniquem entre si).

Outro ponto a destacar é que um processo malicioso, que escreve tuplas invalidas (escrevendo um *fingerprint* não correspondente com a tupla compartilhada) pode ser detectado por outros processos, os quais podem notificar os servidores para que estes ignorem futuras requisições deste processo malicioso (passo 5 do algoritmo 2).

3.5. Aplicando o Esquema em Espaços de Tuplas Tolerantes a Intrusões

Este seção descreve como os protocolos de escrita e leitura, anteriormente apresentados, podem ser aplicados em um espaço de tuplas replicado, usando os dois principais modelos de replicação para tolerância a faltas maliciosas.

3.5.1. Sistemas de Quóruns Bizantinos

O BTS (*Byzantine Tuple Space*) [Bessani et al., 2006b] é a primeira proposta conhecida na literatura para implementação de espaço de tuplas tolerante a faltas maliciosas. Seus algoritmos para inserção e leitura de tuplas (operações out e rdp, respectivamente) são bastante simples e eficientes. A integração dos algoritmos 1 e 2 do esquema de confidencialidade neste espaço é direta, praticamente nenhuma modificação é necessária. Isto ocorre por três motivos básicos: *(i.)* o BTS requer $n \geq 3f + 1$, logo $n - f > f + 1$ e portanto sempre existirão $f + 1$ servidores corretos para retornar fragmentos válidos de uma tupla corretamente inserida; *(ii.)* a inserção de uma tupla no BTS é feita através de um protocolo trivial onde o cliente envia uma mensagem com a tupla a ser inserida para todos os servidores do sistema, desta forma basta que ele envie os diferentes fragmentos (e demais informações associadas a tupla) aos diferentes servidores para inserir uma tupla com confidencialidade; e *(iii.)* a leitura de uma tupla no BTS se dá quando a mesma está presente em pelo menos $f + 1$ servidores, desta forma, com o esquema de confidencialidade, mudamos o algoritmo de leitura para que o cliente considere $f + 1$ tuplas com o mesmo *fingerprint* e que tenham fragmentos válidos.

3.5.2. Replicação Máquina de Estados

A aplicação do esquema de confidencialidade em um espaço de tuplas com este tipo de replicação é relativamente simples e direta, no entanto, alguns cuidados a mais são necessários. Devido a necessidade de acordo entre o estado das réplicas, todas as requisições ao serviço devem ser feitas utilizando uma primitiva de difusão com ordem total [Castro and Liskov, 2002, Schneider and Zhou, 2005]. Desta forma, não é possível enviar diferentes versões de uma requisição para diferentes servidores (contendo apenas

seu fragmento da tupla). A forma de usar o esquema neste modelo é fazer com que o cliente cifre cada um dos fragmentos com a chave secreta compartilhada entre ele e o servidor que vai armazenar esse fragmento. Sendo assim, cada servidor terá acesso apenas ao fragmento a ele endereçado (caso contrário, um servidor faltoso teria acesso a todos os fragmentos e poderia remontar e revelar a tupla).

Apesar de acrescentar um custo adicional no protocolo básico de confidencialidade, o mesmo é bastante aceitável se considerarmos que: *(i.)* apenas criptografia simétrica é usada, e esta é muito rápida se comparada as demais operações criptográficas do esquema; *(ii.)* o número de servidores n não é esperado ser muito grande (uma boa estimativa seria $n \leq 10$), e portanto o número de cifragens no cliente não será grande; e *(iii.)* o servidor realiza apenas uma operação criptográfica a mais para recuperar seu fragmento da requisição, portanto a maior parte do trabalho extra fica no cliente, o que permite ao sistema atender um (potencial) grande número de clientes.

4. Implementação e Ambiente de Execução

Os algoritmos apresentados na seção anterior foram implementados utilizando o arcabouço para prototipação, simulação e execução de algoritmos distribuídos NEKO [Urbán et al., 2002].

Os canais confiáveis e autenticados do sistema são implementados através do uso de *sockets* TCP e chaves de sessão baseadas no algoritmo *HmacSHA-1*. As assinaturas utilizadas nos protocolos são implementadas usando os algoritmos *SHA-1* e *RSA* (com 1024 bits, usada no algoritmo de replicação máquina de estados) para resumos e criptografia assimétrica, respectivamente. Toda implementação se baseia no uso da biblioteca de criptografia do Java 1.5 (*Java Cryptography Extensions*), com exceção do esquema de compartilhamento de segredo verificável publicamente, que foi completamente implementado utilizando a biblioteca de aritmética modular disponível no Java (especialmente a classe `BigInteger`) seguindo as especificações do trabalho original [Schoenmakers, 1999].

O ambiente de testes consta de 5 máquinas com a mesma configuração de *hardware* (AMD Athlon XP 1.9Ghz, 512MB de RAM, placa *ethernet* de 100MB/s) conectadas por um *switch* 1GB/s. O ambiente de *software* em todas as máquinas é também homogêneo: S.O. GNU/Linux *kernel* 2.6.12, máquina virtual Java da SUN versão 1.5.0_06.

5. Resultados e Análises

A fim de avaliar o esquema de confidencialidade proposto, realizamos alguns experimentos com a sua aplicação na replicação máquina de estados baseada no algoritmo PAXOS BIZANTINO [Castro and Liskov, 2002]. Os resultados destes experimentos são apresentados nesta seção. Todos os valores reportados aqui compreendem o tempo médio necessário (em milisegundos) para a execução de uma operação por um cliente do sistema, recolhido a partir de 500 execuções da operação e excluindo-se os 5% dos valores com maior desvio. O tamanho da tupla utilizada nos testes é de 50 bytes, divididos em 4 campos, todos comparáveis. Os valores em tempo referidos no texto a seguir são aproximados.

A tabela 1 apresenta os custos da adição de confidencialidade no espaço de tuplas. Estes custos são apresentados considerando o espaço de tuplas replicado em 4, 7 e 10

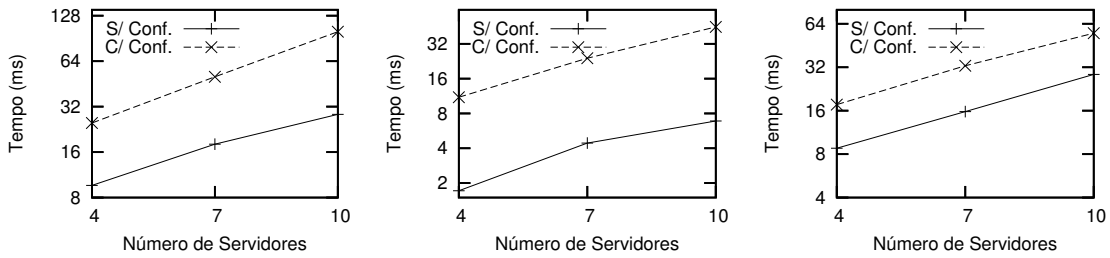
servidores. Analisando esta tabela, podemos perceber que a maior parte do tempo despendido com o mecanismo de confidencialidade esta relacionado com o cliente. Apenas as operações de inserção (OUT) exigem algum aumento do processamento no servidor. Este é um fato importante, pois indica a capacidade de escalar do sistema.

Nº de servidores	4			7			10		
Operação	**OUT**	**RDP**	**INP**	**OUT**	**RDP**	**INP**	**OUT**	**RDP**	**INP**
Custo no cliente	4,5	6,55	6,15	6,06	8,6	7,55	8,48	11,41	10,95
Custo no servidor	8,3	0	0	14,15	0	0	20,13	0	0
Aumentos no custo	12,8	6,55	6,15	20,21	8,6	7,55	28,61	11,41	10,95

Tabela 1. Custo do mecanismo de confidencialidade (valores em ms).

Os custos nas operações de inserção estão relacionados com o tempo gasto para gerar e assinar os fragmentos (cliente) e o tempo gasto para verificar os fragmentos e extrair o fragmento destinado ao servidor (servidor). Nas operações de leitura e remoção, apenas os clientes têm aumento no custo de processamento, o qual está relacionado com o tempo necessário para verificar $f + 1$ fragmentos recebidos dos servidores e combinar estes fragmentos. Como esperado, com o aumento do número de servidores, os custos aumentaram, pois mais fragmentos precisam ser gerados, verificados e combinados.

A figura 2 apresenta a latência média para a execução das três operações, com ou sem confidencialidade, no protótipo de um espaço de tuplas tolerante a intrusões. A confidencialidade foi introduzida neste sistema da forma descrita na seção 3.5.2. Na figura 2(a), são apresentados os resultados referentes a operação de inserção. Como os testes foram realizados em 5 máquinas, nos cenários com 7 e 10 servidores, foi necessário executar mais de um processo na mesma máquina (o que aumenta a latência percebida pelo cliente). Caso os testes fossem realizados com um número maior de máquinas, certamente estes valores seriam menores. Como a operação de inserção é a que tem o maior aumento no custo e é a única que exige aumento no processamento do servidor, os valores de latência apresentados, nos cenários com confidencialidade, cresceram mais do que o esperado com o aumento no número de servidores.



(a) OUT (b) RDP (c) INP

Figura 2. Desempenho na replicação máquina de estados.

Na figura 2(b) e 2(c), temos os resultados referentes as operações de leitura e remoção, respectivamente. Nestes cenários, o crescimento da latência foi linear e man-

teve um padrão, visto que nestas operações apenas o cliente tem um aumento no processamento.

6. Trabalhos Relacionados

Vários trabalhos tem proposto a integração de mecanismos de segurança ao modelo de coordenação por espaços de tuplas visando tanto controle de acessos em nível de espaço (quais processos podem executar operações sobre um espaço de tuplas) quanto em nível de tuplas (quais processos podem ler ou remover uma determinada tupla) [Busi et al., 2003, De Nicola et al., 1998, Vitek et al., 2003]. A principal limitação destes trabalhos é assumir um espaço de tuplas confiável, i.e. que não vai revelar informações a partes não autorizadas. Quando consideramos um sistema tolerantes a intrusões este problema se torna muito mais complexo devido a possibilidade de alguns dos servidores (que implementam o espaço replicado) serem controlados pelo adversário. Até onde sabemos, este é o primeiro trabalho a considerar a confidencialidade em espaços de tuplas replicados.

Tomando em conta toda a literatura relacionada à manutenção de confidencialidade em sistemas de armazenamento tolerantes a faltas Bizantinas, o esquema apresentado neste trabalho guarda alguma semelhança com alguns trabalhos recentes, que fazem uso de códigos de apagamento [Rabin, 1989]. Estes trabalhos tentam otimizar o espaço usado para o armazenamento de dados em sistemas replicados, dispersando as informações (tipicamente arquivos) em diversos servidores. Em particular, o trabalho apresentado em [Cachin and Tessaro, 2006] provê confidencialidade e otimização de armazenamento através do uso de um esquema de criptografia de limiar bastante custoso, executado nos servidores através de um protocolo de difusão confiável (onde os servidores trocam mensagens entre si). O esquema proposto neste trabalho é baseado na eficiente primitiva de compartilhamento de segredo verificável publicamente proposta por [Schoenmakers, 1999], não requer nenhum tipo de comunicação entre os servidores e coloca a maior parte do processamento criptográfico do lado cliente, o que garante a boa escalabilidade do esquema. Como o espaço de tuplas é muito mais um mecanismo de coordenação do que de armazenamento, e as tuplas nele armazenada são usualmente pequenas, o esquema de confidencialidade proposto não leva em consideração a otimização do armazenamento.

7. Conclusões

Este trabalho apresentou uma solução para manutenção de confidencialidade em espaços de tuplas tolerantes a intrusões. Esta solução se baseia no uso de um eficiente esquema de compartilhamento de segredo verificável publicamente, juntamente com resumos criptográficos e criptografia simétrica.

O esquema proposto foi implementado e alguns experimentos foram realizados, demonstrando que, apesar do esquema adicionar um custo adicional no tempo necessário para a execução de uma operação no espaço de tuplas replicado, este custo é relativamente aceitável se comparando a alta complexidade da qualidade de serviço sendo oferecida.

Referências

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

Bellare, M. and Rogaway, P. (1993). Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. of the 1st ACM Conference on Computer and Communications Security*, pages 62–73.

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

Bessani, A. N., Fraga, J., and Lung, L. C. (2006b). BTS: A Byzantine fault-tolerant tuple space. In *Proc. of the 21st ACM Symposium on Applied Computing - SAC 2006*.

Busi, N., Gorrieri, R., Lucchi, R., and Zavattaro, G. (2003). SecSpaces: a data-driven coordination model for environments open to untrusted agents. In *Electronic Notes in Theoretical Computer Science*, volume 68.

Cachin, C. and Tessaro, S. (2006). Optimal resilience for erasure-coded Byzantine distributed storage. In *Proc. of Dependable Systems and Networks, DSN 06*.

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

De Nicola, R., Ferrari, G. L., and Pugliese, R. (1998). Klaim: A kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330.

Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654.

Fraga, J. and Powell, D. (1985). A fault- and intrusion-tolerant file system. In *Proceedings of the 3rd Int. Conference on Computer Security*, pages 203–218.

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

Gelernter, D. and Carriero, N. (1992). Coordination languages and their significance. *Communications of ACM*, 35(2):96–107.

Goldereich, O. (2001). *Fundamentals of Criptography: Basic Tools*, volume 1. Cambridge Press, Cambridge - Massachussets.

Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programing Languages and Systems*, 4(3):382–401.

Rabin, M. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348.

Schneider, F. and Zhou, L. (2005). Implementing trustworthy services using replicate state machines. *IEEE Security & Privacy*, 3(5):34–43.

Schoenmakers, B. (1999). A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Advances in Cryptology - CRYPTO'99*, volume 1666 of *LNCS*, pages 148–164.

Urbán, P., Défago, X., and Schiper, A. (2002). Neko: A single environment to simulate and prototype distributed algorithms. *Journal of Information Science and Engineering*, 18(6):981–997.

Veríssimo, P., Neves, N. F., and Correia, M. P. (2003). Intrusion-tolerant architectures: Concepts and design. In *Architecting Dependable Systems*, volume 2677 of *Lecture Notes in Computer Science*. Springer-Verlag.

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