Luciano Barreto

Controle de Autenticação Tolerante a Intrusões em Federações de *Clouds*

Florianópolis
2017

Luciano Barreto

Controle de Autenticação Tolerante a Intrusões em Federações de *Clouds*

Tese submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do Grau de Doutor em Engenharia de Automação e Sistemas
Orientador: Prof. Dr. Joni da Silva Fraga
Coorientador: Prof. Dr. Frank Augusto Siqueira

Florianópolis
2017

Ficha de identificação da obra elaborada pelo autor
através do Programa de Geração Automática da Biblioteca Universitária
da UFSC.

Barreto, Luciano
Controle de Autenticação Tolerante a Intrusões em Federações de Clouds / Luciano Barreto ; orientador, Joni da Silva Fraga; coorientador, Frank Augusto Siqueira - SC, 2017.
142 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2017.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Controles de Autenticação. 3. Tolerância a Intrusão. 4. Cloud Computing. I. da Silva Fraga, Joni . II. Augusto Siqueira, Frank. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

Luciano Barreto

Controle de Autenticação Tolerante a Intrusões em Federações de *Clouds*

Esta Tese foi julgada adequada para obtenção do Título de Doutor e aprovado em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Florianópolis, 29 de março de 2017.

_____
Prof. Daniel Coutinho, Dr.
Coordenador do Curso

Banca Examinadora:

_____
Prof. Joni da Silva Fraga, Dr.
Orientador
Universidade Federal de Santa Catarina

_____
Prof. Frank Augusto Siqueira, Dr.
Coorientador
Universidade Federal de Santa Catarina

_____
Prof. Altair Olivo Santin, Dr.
Pontifícia Universidade Católica do Paraná

_____
Prof. Carlos André Guimarães Ferraz, Dr.
Universidade Federal de Pernambuco

_____
Prof. Leandro Buss Becker, Dr.
Universidade Federal de Santa Catarina

_____
Prof. Mario Dantas, Dr.
Universidade Federal de Santa Catarina

_____
Prof. Rafael Rodrigues Obelheiro, Dr.
Universidade do Estado de Santa Catarina

AGRADECIMENTOS

Primeiramente agradeço a meus pais, Selma Martins Barreto e Mario Barreto e aos meus irmãos Adriano Barreto e Marcio Ivan Barreto, pelo apoio e atenção. Aos meus irmãos de consideração, Rodrigo Teixeira de Castro (Driguera) e a Joice Gonzales, os quais sempre estiveram ao meu lado, vendo minha trajetória e vibrando com meu êxito.

Agradecer aos amigos de laboratório, Gilmar, André, Tadeu, Lange, e todos os outros que esqueci de numerar aqui e também a todos os novos amigos e pessoas especiais que conheci nessa caminhada.

Ao professor Antonio Puliafito, meu orientador em período de estágio sanduiche, e também todos os amigos italianos que fiz e que continuo tendo contato.

Por fim, agradeço aos meus orientadores prof. Joni da Silva Fraga e prof. Frank Augusto Siqueira, pelo suporte e ajuda nesta caminhada para me tornar pesquisador, e também a CAPES, pelo apoio financeiro, que me permitiu dedicação total a pesquisa.

RESUMO

Federações de *clouds* tem sido um tópico de muitos estudos atualmente na área de Sistemas Distribuídos. A associação de provedores de *clouds* para formação de federações pode ser movida pela necessidade de pequenos provedores se unirem no sentido de competirem em mercado agressivo. O objetivo principal desta tese sempre foi o estudo dos controles de segurança em federações de *clouds*. A autenticação por ser peça fundamental na implantação de políticas de segurança acabou sendo o foco principal de nossos esforços. Em se tratando dos controles de autenticação, um ponto que quase sempre permanece em aberto nas formações destas federações é o gerenciamento de seus usuários. Neste sentido desenvolvemos uma proposta de federação de *clouds* cujas autenticações de seus usuários são baseadas no uso de provedores de identidades. Por se tornar ator principal na aplicação de políticas nestas federações, usamos técnicas que agregassem segurança e tolerância a intrusões a estes provedores de identidades. E, neste sentido, exploramos o uso da virtualização para permitir a proteção de entidades importantes para a segurança destes Provedores de Identidades (*Identity Providers – IdPs*). Estes trabalhos preliminares de autenticação foram incorporados ao projeto *SecFuNet* e adotaram características definidas pelos parceiros de projeto, entre estas, o uso de processadores seguros. Posteriormente, diante de limitações de flexibilidade impostas pelo *hardware* especializado usado no projeto *SecFuNet*, outra abordagem envolvendo o armazenamento de informações de usuários em seus *IdPs* foi desenvolvida. Nesta segunda abordagem, a própria federação de *clouds* através de recursos de memorização foi utilizada como base para o armazenamento de credenciais e atributos de usuários. Protocolos de disseminação baseados em técnicas de compartilhamento de segredo foram introduzidos no modelo, de forma que o armazenamento das informações de usuários se mantivessem seguras mesmo em casos de intrusões tanto nos *IdPs* como em parte dos provedores de *cloud* da federação. A evolução de nossos trabalhos envolveu protótipos, os quais foram submetidos a vários experimentos de testes a fim de verificar sua viabilidade prática de nossas propostas.

Palavras-chave: *Cloud Computing*; Controles de Autenticação; Tolerância a Intrusão.

ABSTRACT

Cloud federations have been the topic of many studies in the area of Distributed Systems. An association of cloud providers for federation formation can be motivated by the need for small providers not to groups their efforts to survive in the aggressive market. The main objective of this thesis is the study of security controls in cloud federations. Authentication as a key part of deploying security policies was the primary focus of our efforts. One point that usually remains open in federations is the management of its users. In this sense, a proposal of federation of clouds whose authentications of its users are based on the use of identities providers. By becoming a key point in policy enforcement in these federations, we use techniques that deploys security and intrusion tolerance to these identity providers. In this sense, we explore the use of virtualization to allow protection of important entities for an IdPs security. These preliminary works on authentication have been incorporated into the SecFuNet project and featured features defined by project partners, among them, and use of secure processors. Subsequently, faced with limitations of flexibility imposed by specialized hardware used no SecFuNet project, another approach involving the storage of information in their IdPs was developed. In this second approach, the federation of clouds was used as memory and database resources for storing attributes of users. Dissemination protocols based on secret sharing techniques for was applied on this model, so that the stored user information has remained secure even in cases of intrusion both in the IdPs and in part the cloud providers of the federation. The evolution of our work involved prototypes, which were submitted to several test experiments.

Keywords: Cloud Computing; Authentication Controls; Intrusion Tolerance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AuS – *Authentication Service* (Serviço de Autenticação)
AgS – *Agreement Service* (Serviço de Acordo)
IdP – *Identity Provider* (Provedor de Identidades)
SLA – *Service Level Agreement* (Acordo em Nível de Serviço)
SP – *Service Provider* (Provedor de Serviços)
RP – *Resource Provider* (Provedor de Recursos)
RD – *Repository of Descritors* (Repositório de Descritores)
XRI – *Extensible Resource Identifier* (Identificador de Recursos)
XRDS – *eXtensible Resource Descriptor Sequence* (Descritor de Sequência de Recursos)

SUMÁRIO

1. INTRODUÇÃO

A associação entre provedores de serviços de *cloud* para o compartilhamento de recursos tem sido apresentado na literatura como solução para otimizar ou complementar o uso destes recursos. Estas associações de *clouds* são caracterizadas por vários termos em diferentes trabalhos: federação de *clouds* (KURZE et al., 2011), *Inter-cloud* (GROZEV; BUYYA, 2012),(KERTESZ, 2014), *Multi-cloud* (GROZEV; BUYYA, 2012) e *Cross-Cloud* (CELESTI et al., 2010a). Embora as diferenças de nomenclatura e mesmo que muitas das propostas para a formação destas redes de confiança entre *clouds* apresentem diferentes focos, o objetivo final é sempre o uso de serviços de diversos provedores de *clouds* para atender a demanda de clientes e os *SLAs* (*Service-Level Agreement*) acordados.

A utilização destas federações em *clouds* permite tipos de compartilhamento ocorrendo em diversos níveis de recursos (máquinas virtuais, armazenamento, redes virtuais, *software*, etc.) e são justificáveis em diferentes situações, tais como: a falta de recursos em um provedor para suportar grandes demandas, serviços ou recursos originalmente não oferecidos por determinado provedor, a distribuição de uma aplicação entre diferentes provedores (definida pelo próprio usuário), colaboração entre pequenos provedores de *clouds*, dentre outros. O interesse nestas associações é plenamente aceitável, porém o agrupamento de provedores de *clouds* nestas redes de confiança que atendem um vasto número de usuários coloca também grandes desafios na autenticação e autorização destes usuários junto aos provedores.

Uma tendência em sistemas distribuídos de larga escala como *clouds*, grades computacionais e redes colaborativas em geral, tanto para a autenticação como na autorização, é o uso de infraestruturas que realizem o gerenciamento de identidades (JØSANG et al., 2005). Estas infraestruturas normalmente integram políticas e tecnologias, permitindo às diferentes organizações que participam destes grandes sistemas terem aplicações que ultrapassem seus domínios locais (domínios administrativos ou de políticas) perfeitamente credenciadas e autorizadas pelas diferentes políticas locais. Com isto, usuários e aplicações podem ter acesso, de maneira segura, a recursos remotos, desde que possuam as credenciais exigidas pelas políticas dos provedores destes recursos.

A infraestrutura necessária para o gerenciamento de identidades envolve muitas vezes um sistema de autenticação e um sistema de gerenciamento de atributos. Estes atributos fornecem informações adicionais (do usuário) para a concretização dos controles de autorização. Em muitas propostas e infraestruturas, as autoridades de autenticação

também concentram o gerenciamento de atributos e são identificados como provedores de identidades (*Identity Providers - IdPs*). Vários modelos de gerenciamento de identidades são identificados na literatura (dentre os quais destacamos o centralizado, identidades federadas (NUÑEZ; AGUDO, 2014) e *user-centric* (JØSANG et al., 2005)) e, em muitos destes modelos, a autenticação não é mais realizada em provedores de serviços (*SPs*), mas sim em autoridades independentes e confiáveis de autenticação (em *IdPs* externos). A autorização, porém, é sempre executada nos provedores de serviço porque estes conhecem seus recursos e suas políticas e devem zelar pelos mesmos.

1.1. MOTIVAÇÃO

Como forma de estender os domínios de autenticação para grandes sistemas, os provedores de identidades (*IdPs*) passaram a ser o ponto central na aplicação de políticas de segurança em *clouds* e federações. Porém, por serem serviços que ficam disponíveis via *Internet*, estes *IdPs* estão sujeitos a ataques que podem resultar em intrusões, o que seria catastrófico para a segurança das informações e recursos. Embora possam ser usados componentes seguros e técnicas que não deixem as informações de usuários disponíveis em um *IdP* invadido, estes cuidados não impedem que *IdPs* possam ser alvo de "implantações" de comportamentos maliciosos. Atuando segundo estes comportamentos, um *IdP* malicioso pode certificar indivíduos não autorizados pelas políticas de segurança (intrusos), assim como também pode não autenticar usuários reconhecidos pelas mesmas políticas.

Diante destas deficiências, a motivação geral desta tese é o desenvolvimento de um sistema de gerenciamento de identidades, baseado no uso de Provedores de Identidade, que ofereça alto grau de segurança no que se refere ao armazenamento de credenciais e atributos de usuários. Além da segurança destas informações, e por se tratar de um serviço essencial para provedores de *clouds* e federações, os *IdPs* devem manter o seu funcionamento correto mesmo que intrusões ocorram e tentem afetar o funcionamento do serviço.

1.2. OBJETIVOS E PERCURSOS DE NOSSOS TRABALHOS

O objetivo geral desta tese de doutorado foi o estudo dos controles segurança federações de *clouds*. A autenticação por ser peça fundamental na implantação de políticas de segurança é o foco principal de nossos esforços. Diante disto, um sistema de autenticação foi construído com o propósito de manter as propriedades de segurança do mesmo.

A dimensão destes novos ambientes computacionais exige muito mais que a manutenção destas propriedades de segurança. É necessário que este sistema de autenticação possa se adequar a possíveis ações maliciosas tentadas contra os mesmos. Portanto, o uso de mecanismos de tolerância a intrusões estava em nossas perspectivas e se justificam pela disponibilidade destes serviços na *Internet*. Outros aspectos podem ser incluídos para justificar a tolerância a intrusões como, por exemplo, a disponibilidade de informações de usuários em ambientes que são essencialmente compartilhados.

Para desenvolver este sistema de autenticação foi necessário um entendimento do que é uma federação de *clouds*. Desenvolvemos então um modelo de federação que serviu de veículo para nossos estudos permitindo que tivéssemos uma ideia perfeita do funcionamento dos controles de autenticação e de autorização nestes ambientes.

O modelo de autenticação foi primeiramente desenvolvido dentro de um projeto Brasil-Europa e era condicionado ao uso de componentes especiais. Este modelo de autenticação foi desenvolvido e devidamente testado. Mas, ficaram evidentes para nós as dificuldades com a flexibilidade do modelo.

Queríamos então dotar nossas proposições de autenticação de características de flexibilidade e de escalabilidade. Outra preocupação que se fez foi a de como desenvolver um sistema de armazenamento na federação que permitisse o controle seguro de credenciais e atributos de usuários. Lembrando que estes ambientes são essencialmente compartilhados e gerenciados por terceiros. Diante destas questões levantadas, acrescentamos ao nosso modelo de autenticação um novo conjunto de protocolos e conceitos de modo a preencher nossas expectativas de segurança no armazenamento das informações de usuário e ao mesmo tempo perseguindo propriedades de flexibilidade e escalabilidade.

Nas evoluções que tivemos em nossos trabalhos sempre perseguimos análises quantitativas que nos dessem uma ideia de comportamento de nossos protótipos. Nestas análises muitas vezes nos utilizamos de serviços de *clouds* que nos permitissem o levantamento das características de nossos protótipos diante da distribuição geográfica destes serviços. Isto nos deu uma ideia do funcionamento de nossas propostas e do que poderia impactar em seus funcionamentos em termos de tempo de resposta, caracterizando ambientes reais nestas formações de federação com provedores em diferentes regiões pelo mundo.

1.3. ORGANIZAÇÃO DA TESE

Esta tese está organizada em seis capítulos. Neste capítulo foi descrito o contexto geral do trabalho desenvolvido, assim como a motivação e os objetivos a serem atingidos. O capítulo 2 introduz conceitos utilizados no desenvolvimento de nossas soluções. O capítulo 3 apresenta nossa proposta de federação de *clouds*, a qual descreve um modelo que agrega recursos de vários provedores de *cloud*. Incluímos nesta proposta o conceito de gerenciamento de identidades como base para os controles de autenticação e de autorização.

O capítulo 4 apresenta nossa proposta de provedor de identidades tolerante a intrusão baseado em virtualização, onde exploramos ferramentas e tecnologias para alcançar os requisitos desejáveis para este tipo de sistema. No capítulo 5 exploramos a extensão do modelo desenvolvido no Capítulo 4 de forma a distribuir os dados (credenciais e atributos) de usuários em *clouds* da federação proposta no Capítulo 3, utilizando para isso algoritmos de compartilhamento de segredo.

Por fim, no Capítulo 6 apresentamos as conclusões obtidas no desenvolvimento desta tese assim como as perspectivas de trabalhos futuros.

2 CONCEITOS

Este capítulo apresenta conceitos de Segurança Computacional, Gerenciamento de Identidades e de Computação em Nuvem. Estes conceitos são apresentados de forma que favoreçam a base para compreensão dos trabalhos descritos nesta tese.

As seções 2.1 e 2.2 introduzem os conceitos básicos de segurança computacional, além de apresentar os mecanismos usuais para a segurança em sistemas distribuídos. A seção 2.3 explora com mais detalhes os mecanismos usados em sistemas distribuídos. Com isto, são apresentados os principais conceitos sobre os controles de autenticação e de autorização em sistemas distribuídos. Colocamos também nesta seção 2.3 os conceitos de gerenciamento de identidades como parte dos controles de autenticação. Por fim, na seção 2.4 são apresentados os principais conceitos e modelos de computação em nuvem.

2.1 CONCEITOS DE SEGURANÇA COMPUTACIONAL

Nos dias de hoje, devido ao crescente uso de sistemas computacionais em nossas vidas, a segurança destes sistemas se tornou um requisito de qualidade de serviço que se faz indispensável. A grande disseminação de dispositivos computacionais embarcados e móveis espalhados por todas as áreas e locais da sociedade moderna comprovam esta presença citada. A conectividade destes sistemas é também global, através do uso da Internet. Com isto, no armazenamento controlado de informações e nas comunicações passam a ser determinantes requisitos de segurança. A evolução dos computadores e dispositivos em geral fez com que o significado, a granularidade e as implicações da segurança computacional tenham mudado no decorrer de todos estes anos (LANDWEHR, 2001)(LANDWEHR, 2014).

Na sequência introduzimos conceitos de segurança necessários para o entendimento deste texto.

2.1.1 Segurança Computacional

> "A segurança corresponde a uma qualidade de serviço que é obtida através de meios, técnicas e mecanismos que visam à manutenção das propriedades de *confidencialidade*, *integridade* e *disponibilidade* de informações e recursos em um sistema computacional" (BRINKLEY; SCHELL, 1995).

Às propriedades de segurança citadas, muitos autores ainda acrescentam as de *autenticidade* e de *não repúdio* (LANDWEHR, 2001).

2.1.2 Políticas de Segurança

O termo *política de segurança* pode ser abrangente ou assumir significados mais específicos, dependendo do nível em que se considera. Se considerarmos a administração de uma instituição, a política de segurança pode tomar a forma de um conjunto de regras e práticas que regulamentam como a instituição gerencia, protege e distribui suas informações. Esta ótica deve se refletir também nos ambientes computacionais, onde *política de segurança* passa a ser definida como *um conjunto de regras* que especificam como um sistema provê os seus serviços, mantendo as propriedades de confidencialidade, integridade e de disponibilidade das informações e recursos que controla (SHIREY, 2007).

As políticas de segurança em sistemas computacionais são classificadas em duas categorias: as *discricionárias* e as *obrigatórias*. Nas discricionárias, os acessos a cada recurso ou informação são controlados livremente pelo proprietário ou responsável do mesmo, segundo a sua vontade. Em relação às políticas obrigatórias (não discricionárias ou mandatórias), as autorizações de acesso são definidas através de um conjunto incontornável de regras globais que expressam algum tipo de estratégia ou regulamento de uma organização, envolvendo a segurança das informações no sistema como um todo (LANDWEHR, 2001).

2.1.3 Principais e Intrusos

As regras definidas pela política de segurança determinam as entidades autorizadas e responsáveis pelas ações executadas sobre informações ou recursos mantidos no sistema, normalmente identificadas como *principal* (ou sujeito autorizado pela política). Um principal pode ser um usuário, um processo ou ainda uma máquina em uma rede de computadores. A entidade (usuário, processo ou máquina) não autorizada pela política que ganha acesso a recursos de um sistema computacional é denominado de *intruso* (sujeito não autorizado).

2.1.4 Modelos de Segurança

Os modelos de segurança correspondem a descrições formais do comportamento de um sistema atuando segundo regras de uma política de segurança. Estes modelos são representados na forma de um conjunto de entidades e relacionamentos (GOGUEN; MESEGUER, 1982). Os modelos de segurança para controle de acesso se apresentam na literatura divididos em três tipos básicos (SANDHU; SAMARATI, 1996): baseados em identidade[1] ou discricionários (DAC - *Discretionary Access Control*); baseados em regras ou obrigatórios (MAC - *Mandatory Access*

[1] *OSI Security Architecture Standard ISO/IEC* 7498-2

Control); e os baseados em papéis (*RBAC - Role-Based Access Control*). Outros modelos mais recentes têm sido introduzidos na literatura, como o $UCON_{ABC}$ (PARK; SANDHU, 2004) e o *RADAC* (KANDALA; SANDHU; BHAMIDIPATI, 2011).

2.1.5 Violações de Segurança

As *violações de segurança* em sistemas computacionais correspondem a não atender de alguma forma a política de segurança, de modo que não se verifiquem uma ou mais propriedades de segurança, citadas anteriormente. Note que a não verificação de cada uma destas propriedades dá origem a uma das violações de segurança possíveis em um sistema. Ou seja, a não verificação da propriedade de confidencialidade dá origem à violação *revelação não autorizada de informação*; quando a integridade não é verificada temos então a violação de *modificação não autorizada*; se informações ou serviços de um sistema se tornam inacessíveis (não disponíveis) aos seus usuários (principais do sistema) ocorre o que é identificado na literatura como a violação de *negação de serviço*.

2.1.6 Vulnerabilidades, Ameaças e Ataques

Antes de expor as principais violações em ambiente distribuído, algumas definições devem ser feitas. Entende-se por *vulnerabilidades* (*vulnerabilities*) fraquezas ou imperfeições em implementações ou procedimentos de serviços e sistemas, oriundas de falhas de concepção ou de configuração dos mesmos e que expõem os recursos de um sistema computacional a ataques. Uma *ameaça* (*threat*) é caracterizada por um conjunto de ações possíveis que explorariam as vulnerabilidades (circunstâncias, condições, conhecimento sobre o sistema) e colocariam em risco as propriedades de segurança. Uma ameaça, quando posta em ação, ou mesmo ações que visam explorar vulnerabilidades são identificadas como *ataques* (*attacks*) à segurança do sistema.

Os ataques à segurança de um sistema computacional distribuído se dão normalmente através de uma rede (subsistema de comunicação) pela qual transitam informações na forma de mensagens fluindo de uma fonte, como um arquivo ou uma região da memória, para um destino remoto. Em *Lendwher* (LANDWEHR, 2001) é apresentada uma classificação amplamente utilizada, onde os ataques em um sistema distribuído são divididos em passivos e ativos.

a) Ataques passivos

Os ataques passivos são fundamentalmente baseados em escutas (monitoramento das comunicações) com o objetivo de obter informações transmitidas no sistema. Este tipo de ataque é de difícil detecção, pois não

envolve qualquer alteração nos dados, então, é mais importante a prevenção do que a detecção. Os ataques passivos são subdivididos em:

- *Visualização do conteúdo da mensagem*: uma mensagem pode conter informações sensíveis ou confidenciais. A necessidade de prevenir que o atacante possa capturar o conteúdo dessas transmissões leva normalmente ao uso de técnicas criptográficas.
- *Análise de tráfego*: Mesmo quando a criptografia é usada para proteger o conteúdo (*payload*) da mensagem, as informações de cabeçalho (*header*) são deixadas em claro para permitir o roteamento. Estas informações podem ser usadas para inferir situações entre os pares comunicantes ou podem ser usadas para ataques subsequentes, como a negação de serviço, visando prejudicar os pares comunicantes. As análises de tráfico são de difícil solução.

b) Ataques ativos

Este tipo de ataque envolve a modificação ou criação de fluxos de dados no suporte de comunicação:

- *Personificação* (*masquerade*): o atacante tenta se fazer passar por um usuário válido do sistema e assumir com isto os seus privilégios.
- *Repetição de mensagens* (ataques de *replay*): consiste na captura de mensagens (ataque passivo) e a posterior retransmissão das mesmas. O atacante espera com estas ações obter vantagens no sistema. Por exemplo, um atacante captura e retransmite instruções de transferência de fundos para uma conta bancária sob o controle do atacante.
- *Modificação de mensagens* (*modification of messages*): o atacante modifica parte de uma mensagem legítima do sistema para produzir um efeito de autorização.
- *Ataque de negação de serviço* (*Denial of Service*): o objetivo deste tipo de ataque é tornar indisponível ou provocar o mau funcionamento de um determinado serviço. Por exemplo, um atacante pode inundar determinado serviço com solicitações, fazendo com que o serviço não consiga atender a solicitações legítimas do sistema.

Embora os ataques passivos sejam de difícil detecção, ainda assim há meios disponíveis de prevenção (uso da criptografia, por exemplo). Por outro lado, a detecção é sempre possível para ataques ativos, mas para sua prevenção total é necessária a proteção física do suporte de comunicação durante todo o tempo. Se considerarmos a Internet, isto se torna praticamente impossível.

Muitos tipos de ataques e ameaças são identificados na literatura, mas o sucesso dos mesmos, na maioria das vezes, está condicionado à existência de vulnerabilidades de segurança em sistemas. O CERT[2] (Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil) fornece várias informações em seu site sobre incidentes de segurança.

2.2 MECANISMOS DE SEGURANÇA

As políticas de segurança, expressas muitas vezes por modelos de segurança, são implementadas através de *mecanismos de segurança*. Para viabilizar a implantação de tais políticas, estes mecanismos são construídos a partir de controles de acesso e controles criptográficos.

2.2.1 Controles de Acesso

Os chamados controles de acesso são implementados por mecanismos que tomam o nome de *monitor de referências* que basicamente implementam as regras de acesso definidas pelas políticas de segurança (LAMPSON, 1971)(KHAN, 2012). A noção de monitor de referências se faz presente nos vários níveis de um sistema. Referências a segmentos de memória são validadas nas camadas inferiores do sistema, envolvendo o *hardware* e, por sua vez, o sistema operacional, através de um serviço de arquivos, valida o acesso a arquivos.

O monitor, como responsável por intermediar todas as requisições de acesso a objetos de um sistema, deve ter algumas propriedades: deve ser inviolável, incontornável (em qualquer requisição de acesso) e pequeno o suficiente para permitir a validação de sua correção. A noção de *núcleo de segurança* foi definida em (LANDWEHR, 1983) como o conjunto de recursos de *hardware* e *software* que permitem a concretização de um monitor de referências.

Estes "monitores de referência" se fazem presente em mecanismos de controle nos modelos discricionário (DAC – *Discretionary Access Control*), obrigatório (MAC – *Mandatory Access Control*), baseado em papéis (*RBAC - Role-Based Access Control*) e em outros, seguindo os modelos de políticas que devem implementar.

2.2.2 Controles Criptográficos

Os controles criptográficos fornecem a base para a maioria dos mecanismos de segurança. A cifragem e decifragem de informações envolvem duas operações básicas: encriptação (*encryption*), processo em que a informação é codificada para ocultar seu significado e decriptação (*decryption*), aplicada à informação codificada para revelar o seu conteúdo. Para executar os controles criptográficos há muitos algoritmos

[2] http://www.cert.br/

disponíveis, sendo baseados em segredos, normalmente chamados de *chaves*. Esses algoritmos são divididos em duas classes: a dos simétricos e a dos assimétricos (ou algoritmos de chave pública).

Algoritmos simétricos usam uma mesma chave para a encriptação e a decriptação dos dados. A confidencialidade das informações depende, neste caso, do segredo desta chave (dita chave secreta em criptosistemas simétricos). Já os algoritmos assimétricos usam uma chave pública para encriptação e uma chave privada para decriptação. Neste caso, as chaves públicas podem ser distribuídas livremente, enquanto as chaves privadas devem ser mantidas em sigilo por seu proprietário. Em relação ao tempo de processamento, os algoritmos assimétricos são mais lentos que os simétricos, porém levam vantagem no aspecto de distribuição das chaves públicas.

Os algoritmos assimétricos ainda agregam propriedades na possibilidade da verificação da autenticidade das informações. Ao cifrar uma informação com a sua chave privada, um principal associa a mesma unicamente a si como fonte geradora da mesma. Este tipo de procedimento é a base do que é chamado de *assinatura digital*. Como somente o principal que originou esta informação cifrada possui esta chave privada, qualquer receptor pode verificar a autenticidade da informação usando a chave pública correspondente, que é disponibilizada a todos os interessados.

Através dos controles criptográficos, de uma maneira geral, é possível alcançar as propriedades de integridade, confidencialidade e autenticidade. A cifragem de informações garante a confidencialidade das mesmas. A autenticidade destas informações é verificada com o uso de técnicas de assinatura digital em *resumos* obtidos através da aplicação de algoritmos específicos (funções de *hash*) destas mensagens ou informações. Estes resumos assinados usados na autenticação das mensagens servem também na verificação da integridade das informações.

O uso de chaves públicas inclui problemas relacionados à origem destas, isto porque qualquer participante pode enviar sua chave pública para outros participantes. Esta facilidade permitiria que usuários mal-intencionados se anunciassem com identidade de um usuário, mas fornecessem suas chaves públicas e, com isto passassem a receber mensagens para as quais não seriam os destinatários por direito. Uma solução para este problema é o uso pelas partes comunicantes de uma entidade confiável, normalmente chamada de Autoridade Certificadora (*AC*), que tem como função assinar com sua chave provada as chaves públicas concatenadas com os nomes de seus proprietários, formando o

que são identificados como certificados. Estes certificados são aceitos devido as relações de confiança da *AC* com as partes comunicantes e a validação destes certificados determina a necessidade da disponibilidade da chave pública da *AC*.

As *ACs* em conjunto com mecanismos de criptografia assimétrica estabelece o que é chamado de sistema de certificação digital ou de Infraestrutura de Chaves Públicas (ICP ou ainda, *PKI* em Inglês). Tipicamente nestas infraestruturas existem dois tipos de certificados: um que associa uma chave pública a um nome (certificados de nome) e outro que liga atributos à chave pública (certificados de autorização). Atualmente vários sistemas de *PKIs* estão em uso, alguns deles patenteados, como o *Pretty Good Privacy* (PGP) (ZIMMERMANN, 1995), e outros certificados específicos, como certificados usados no SET (*Security Electronic Transaction*). O sistema de certificados mais amplamente aceito é o que segue o padrão X.509 (HOUSLEY et al., 2002). Há ainda propostas, como o SPKI/SDSI (ELLISON, 1999) que visam reduzir a complexidade imposta pelo padrão X.509.

2.2.3 Controles Adicionais de Segurança

Além dos controles citados, outros mecanismos de segurança envolvem ações, técnicas, procedimentos ou dispositivos que têm como propósito implantar políticas de segurança. Estes controles (internos), que não atuam diretamente nas requisições de acesso, podem estar presentes nos sistemas. Entre estes estão:

- A *auditoria de segurança*, que corresponde à inspeção independente (por terceiros) dos procedimentos e registros do sistema com intuito de verificar adequação da política de segurança e as possíveis violações do sistema.
- A *detecção de intrusão*, que usa os mesmos registros das auditorias em métodos automatizados de análise em tempo execução com o intuito de identificar atividades anormais no sistema.

Mecanismos que façam uso de técnicas de *backups*, replicações e que permitam recuperar o sistema em situações onde as violações não puderam ser evitadas completam os controles adicionais.

Outros controles, estes externos ao sistema, necessitam ser adicionados aos internos já descritos neste documento. Por exemplo, é necessário que se leve ao conhecimento de cada usuário suas atribuições e responsabilidades, para que esse saiba o que está autorizado a fazer. Se este não estiver treinado e convencido da importância da segurança no ambiente computacional, as demais medidas podem se tornar ineficazes.

2.3 SEGURANÇA EM SISTEMAS DISTRIBUÍDOS

São muitos os mecanismos e controles de segurança usados em sistemas distribuídos e redes de computadores em geral. Neste tópico nós vamos nos ater a dois tipos de controles e mecanismos que são centrais nestes sistemas para a manutenção da segurança dos mesmos: a autenticação e a autorização.

A *Autenticação* em sistemas distribuídos envolve mecanismos que buscam verificar a autenticidade de usuários ou mesmo de informações, relacionando as mesmas com seus principais. Para tanto, um usuário, ao acessar um serviço, deve apresentar credenciais, de forma que seja possível a sua identificação pelo mesmo. Diversos tipos de credenciais podem ser utilizados para a autenticação de um usuário, tais como usuário/senha, protocolos de *challenge*/*response*, certificados digitais, dentre outros.

Em um sistema distribuído, qualquer comunicação ou acesso deve envolver algum suporte de autenticação de modo a assegurar a autenticidade das partes interagindo e das informações trocadas entre as mesmas. Em outras palavras, um serviço de autenticação deve promover a identificação de usuários, prover meios para a autenticação mútua entre partes comunicantes e a garantia da autenticidade dos dados transmitidos entre estas partes.

Os controles de *Autorização* (ou mecanismos de controle de acesso), por sua vez, envolvem a verificação da adequação dos acessos a recursos no sistema distribuído. Esta adequação está ligada à verificação das regras de acesso ao recurso definidas pelas políticas do sistema. Dependendo do tipo de política usado no sistema, permissões são definidas para os diversos principais que acessam serviços ou recursos no sistema.

A implementação de mecanismos para os controles de autenticação e de autorização em sistemas distribuídos é uma tarefa árdua. Estes controles são totalmente afetados pela escalabilidade, se consideramos sistemas complexos. Em outras palavras, a habilidade de um usuário em interagir com muitos serviços disponíveis em grandes sistemas, dependendo dos tipos de políticas definidos nestes sistemas, pode envolver a necessidade de várias credenciais e atributos e de vários mecanismos de controle.

Na literatura são apresentadas algumas abordagens para implementação dos mecanismos de autenticação e autorização. A abordagem tradicional é que cada provedor deve implementar os seus controles de autenticação e de autorização. No entanto, uma tendência forte nos dias atuais é a de uma autenticação centralizada e a autorização

descentralizada mantida sobre cada serviço ou recurso. Esta centralização da autenticação é feita por uma entidade confiável em um domínio e as autorizações são exercidas pelos próprios provedores de serviços deste domínio de políticas.

2.3.1 Gerenciamento de Identidades

O conceito de gerenciamento de identidades está estritamente relacionado aos controles de autenticação que verificam a autenticidade de usuários em sistemas distribuídos. De maneira geral, o gerenciamento de identidades consiste em manter e gerenciar identidades digitais que são representações de entidades reais em sistemas computacionais (CHADWICK, 2009). Cada usuário real do sistema é representado através de identidade virtual, composta por credenciais e atributos que permitem verificar a sua autenticidade e servir como base para as verificações de políticas de autorização.

Em (CHADWICK, 2009) é definido o gerenciamento de identidades como um conjunto de funções e habilidades que visam a administração de informações de modo a garantir a identidade de usuários em um sistema.

Diferentes abordagens de sistemas de gerenciamento de identidades são classificadas na literatura (JØSANG et al., 2005)(RECORDON; REED, 2006). A mais simples e amplamente adotada destas abordagens é a identificada como *modelo tradicional* (Figura 1), onde tanto os controles de acesso (autorização) a um serviço oferecido quanto a autenticação (gerenciamento de identidades) são implantados e executados no provedor do serviço. Este modelo tradicional implica que o usuário deve manter credenciais de acesso para todos os provedores cujos serviços deseja utilizar.

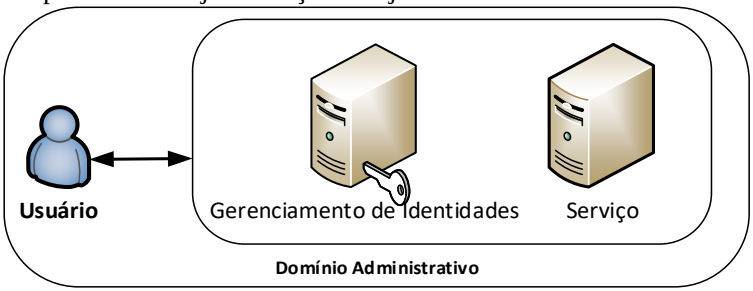


Figura 1 – Modelo de Gerenciamento de Identidades Tradicional

O *modelo centralizado* (Figura 2), que nos últimos anos vem sendo amplamente difundido, separa as preocupações de autorização do serviço com as dos controles de autenticação de seus usuários, ou seja, retira o

gerenciamento de identidades dos provedores de serviço. Neste modelo, as funções de autenticação de usuários são mantidas em máquinas especiais (*Provedores de Identidades - IdPs*) que executam estas funções para todos os provedores de serviço do domínio de nomes considerado. Com isso, os prestadores de serviços mantêm as suas preocupações voltadas para os seus serviços e para as verificações de autorização em relação aos acessos a estes serviços. Com a adesão a esta abordagem, o usuário não precisa mais manter múltiplos registros com suas identidades digitais e credenciais para os vários provedores de serviço. A infraestrutura *OpenID* (OPENID, 2007) é um exemplo de implementação deste modelo.

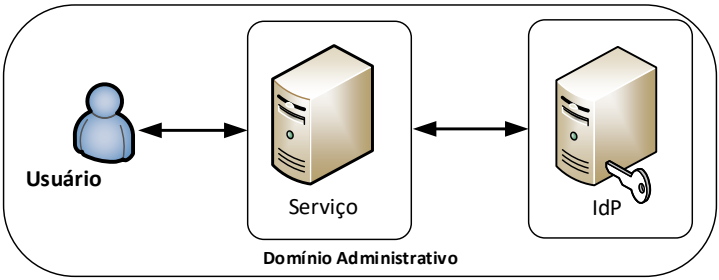


Figura 2 - Modelo de Gerenciamento de Identidades Centralizado

Nesta abordagem centralizada, um provedor de identidades (*IdP*) que mantém todas as credenciais e atributos de usuários do domínio, envia asserções de autenticação e atributos dos usuários autenticados aos provedores de serviço, para que estes executem seus controles de autorização.

Outra grande vantagem deste modelo é a possibilidade de se utilizar o conceito de autenticação única, o *Single Sign-On* (*SSO*), que permite que uma mesma autenticação de um usuário possa ser compartilhada simultaneamente por diversos provedores de serviço. Deste modo, o usuário se autentica uma única vez, através de seu provedor de identidades, e pode ter acesso a serviços de provedores distintos no seu domínio de nomes, durante sua sessão de computação. Ou seja, o usuário não mais é obrigado, ao acessar vários provedores de serviço, a fazer um *login* em cada provedor de serviços que venha a acessar em sua sessão de computação. No processo de autenticação de usuários, o *IdP* emite asserções de autenticação que são usadas quando das verificações de autorização nas requisições de acesso do correspondente principal em provedores de serviço.

Mas, neste modelo centralizado, mesmo com as facilidades na redução do número de contas que um principal necessita manter junto a

diferentes provedores, os serviços que estejam fora do domínio de nomes do usuário não aceitariam estas asserções de autenticação. Ou seja, estes provedores não manteriam relações de confiança com o *IdP* e, portanto, não aceitariam estas asserções.

O *modelo federado* (Figura 3) estende as características do modelo centralizado, fazendo com que os problemas de escala sejam atendidos. Ou seja, no modelo federado, cada domínio local continua com o seu provedor de identidades mantendo as informações de seus usuários e emitindo asserções para seus provedores de serviço. Mas, para que haja a transposição destas asserções entre domínios, no modelo federado, os provedores de identidade estabelecem relações de confiança, permitindo com isto que usuários autenticados em um domínio, possam ter acesso a provedores de serviço de outros domínios usando a autenticação (ou *login*) realizada em seu domínio. A existência destas redes de confiança entre *IdPs* de diferentes domínios caracteriza o que é chamado de *federação de identidades*.

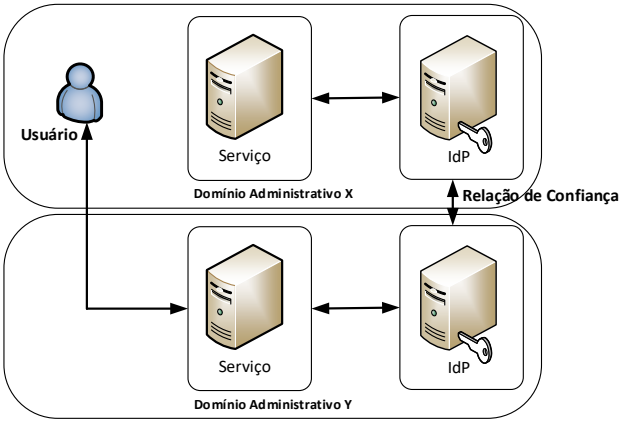


Figura 3 - Modelo de Gerenciamento de Identidades Federadas

As relações de confiança entre *IdPs* nada mais são que acordos entre administradores de *IdPs* que garantem que *tokens* (ou asserções) de autenticações emitidas em um domínio sejam aceitas em outros domínios por provedores de serviços (CAMENISCH; PFITZMANN, 2007). Dessa forma, o modelo de identidades federadas estende o modelo centralizado no sentido de que um usuário possa utilizar serviços de diferentes domínios sem a necessidade de manter contas nestes domínios distintos.

Apesar de existirem restrições ao modelo de identidades federadas, principalmente devido a certa disseminação de atributos de usuários (colocando preocupações em relação à privacidade dos mesmos), o

interesse em sua utilização vem sendo crescente com a proposição de várias infraestruturas que adotam este modelo. As infraestruturas *Liberty Alliance* (LIBERTY, 2003) e *Shibboleth* (MORGAN; SCAVO, 2005) implementam o modelo de identidades federadas.

As preocupações no modelo de gerenciamento de identidades federadas (e que podem ser estendidas também para o modelo centralizado) em relação à privacidade das informações de usuário (nome, senha, dados pessoais, financeiros) fizeram com que outras propostas devolvessem o controle destas informações aos próprios usuários. Estas propostas definem controles centrados no usuário na liberação de seus atributos (JØSANG; POPE, 2005) (BHARGAV-SPANTZEL et al., 2007).

Estes modelos de gerenciamento incluem consultas ao usuário sobre a liberação de seus atributos para provedores de serviço. Na Figura 4 o usuário tenta o acesso ao provedor de serviço, que solicita uma asserção de autenticação do usuário e atributos do usuário para o controle de acesso. O usuário então usa seu *smartcard*, que possui o registro de seus atributos para liberar os mesmos via seu *IdP* para o provedor de serviços (ECLIPSEFOUNDATION, 2010) (CHAPPELL, 2006).

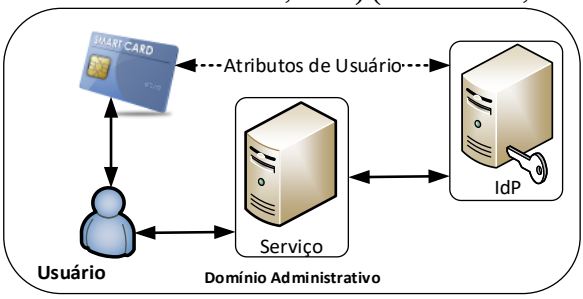


Figura 4 – Gerenciamento de Identidades Centrado no usuário

2.3.2 *Frameworks* e Especificações para Provedores de Identidades

O foco principal desta seção está na descrição dos principais *frameworks* e soluções para a criação de federação de provedores de identidades. Os trabalhos referenciados sobre estes *frameworks* são geralmente especificações que descrevem como estes serviços devem ser implementados assim como os padrões adotados (comunicação, troca de informações, gramática XML, etc.).

a) *SAML* (Security Assertion Markup Language)

O *Security Assertion Markup Language* (*SAML*) (RAGOUZIS et al., 2007) é um conjunto de especificações para a troca de asserções de autenticação, autorização e atributos dentro de uma federação. O principal uso do *SAML* envolve o *Single Sign-On* (SSO), Gerenciamento de Sessões e segurança em aplicações *Web*. Em sua versão mais atual (*SAML* 2.0), as especificações *SAML* são baseadas no *Liberty Alliance Identity Federation Framework* (ID-FF 1.2) (WASON, 2003). As especificações do *SAML* são divididas em Protocolos, Asserções e Perfis, que serão descritos abaixo.

Protocolos e Asserções SAML

Em *Assertions and Protocols for the SAML 2.0* (OASIS, 2005a) é definida a gramática usada nas asserções em seus arquivos XML (*SAML*) e os protocolos de trocas de mensagens entre as diferentes entidades em suas sessões de autenticação. As asserções *SAML* especificam um formato de representação de informações de segurança sobre um sujeito, uma organização, um computador, uma pessoa, dentre outros[3]. Essas informações são validadas por uma entidade chamada *asserting party* ou *SAML authority*, que comprovam as informações emitidas sobre um sujeito.

Os protocolos introduzidos nas especificações *SAML* são definidos em duas camadas: a camada superior, compostas pelos XML *schemas* das mensagens, e a camada inferior, consistindo de especificações que descrevem como são usados os protocolos de mais baixo nível (SOAP, HTTP, etc.) no transporte dessas mensagens *SAML*. Esta separação

[3] Asserções *SAML*, geralmente são compostas das seguintes informações (*tags*) em seus arquivos XML: i) Elemento raiz *SAML:Assertion*, que contém a definição do espaço de nomes, a versão do *SAML* e a data em que a asserção foi emitida por um *SAML authority*; ii) Elemento *SAML:Issuer*, indicando qual autoridade emitiu a asserção. iii) Elemento *SAML:Subject*, que indica o sujeito da asserção; iv) Elemento *SAML:Conditions*, indicando o período de validade da asserção; v) Elemento *SAML:AuthnStatement*, representando a declaração de autenticação. Esta declaração indica a data em que o usuário efetuou sua autenticação e se foi utilizado algum mecanismo de segurança (canais SSL, por exemplo). Além disso, o método de autenticação utilizado pelo usuário (usuário/senha, certificados X.509, *Kerberos*, etc.) é especificado neste elemento.

permite aplicações que operem sobre diversos níveis (camadas) e ferramentas, fazendo uso das asserções *SAML*.

Na camada superior, citada acima, são introduzidos os protocolos indicados na Tabela 1. A camada inferior, por sua vez, define os mapeamentos indicados na Tabela 2.

Tabela 1 – Protocolos da Camada Superior do *SAML*

Protocolos	**Descrição**
Protocolo para Pedidos de Autenticação	Este protocolo define como um sujeito irá requisitar uma nova asserção de autenticação e, opcionalmente, asserções de atributos.
Protocolo de Pedidos e Consulta de Asserção	Protocolo que descreve consultas para obtenção de asserções de autenticação (novas ou já emitidas).
Protocolo de Encerramento de Sessão	Determina as trocas necessárias para que uma sessão de autenticação seja encerrada e os recursos utilizados pelo usuário através dessa asserção sejam liberados de forma apropriada.
Protocolo de Resolução de Artefatos	Define mecanismos para a troca de mensagens SAML. Um identificador de tamanho único, chamado de artefato, é utilizado pelas entidades de autenticação com o objetivo de referenciar as asserções emitidas.
Protocolo de Gerenciamento e Mapeamento de Identificadores de Nomes	Descreve os relacionamentos entre identificadores de valores e sujeitos.

Tabela 2 – Mapeamento definidos na Camada Inferior do SAML

Mapeamentos	**Descrição**
SAML over SOAP	Mapeamento que descreve como mensagens SAML devem ser transportadas em envelopes SOAP através do protocolo HTTP.
HTTP Redirection	Especifica como mensagens SAML serão transportadas em mensagens de redirecionamento HTTP.
HTTP POST	Define como as mensagens SAML serão transportadas em formulários quando codificadas em *base-64*.
HTTP Artifact	Determina a forma em que um artefato é transportado utilizando o protocolo HTTP, sejam em formulários ou em URLs.
SAML URI	Descreve os relacionamentos entre identificadores de valores e sujeitos.

Uma das grandes vantagens de se utilizar provedores de identidades (*IdPs*) que façam a autenticação independentes de provedores de serviços é a facilidade apresentada para o processo efetuar a autenticação única, também chamada de *Single Sign-On* (*SSO*). No *SSO*, um usuário autenticado uma vez pode fazer uso de diversos serviços sem a necessidade de informar novamente suas credenciais.

Em *Profiles for the OASIS Security Assertion Markup Language* (*SAML*) *V2.0* (OASIS, 2005b) são descritos os protocolos de autenticação (*login* de usuário), assim como o protocolo de encerramento de sessão. O protocolo de autenticação é apresentado em seus passos na Tabela 3.

Tabela 3 – Protocolo de Autenticação SAML

Protocolo de Autenticação no SAML	
Passo	***Descrição***
1	O cliente requisita acesso a algum recurso pertencente a um Provedor de Serviços através de uma solicitação HTTP;
2	O Provedor de Serviços responde ao cliente enviando uma requisição de autenticação (**<AuthnRequest> message**);
3	O Cliente emite **<AuthnRequest>** para o Provedor de Identidades utilizado as especificações SAML SOAP;
4	O Provedor de Identidades identifica o cliente;
5	O Provedor de Identidades emite uma mensagem **<Response>** para o cliente utilizando as especificações SAML SOAP, destinada ao Provedor de Serviços;
6	O cliente retransmite esse **<Response>** para o Provedor de Serviços;
7	Baseado nas informações desse **<Response>** que embute uma asserção de autenticação *SAML* assinada pelo *IdP* em seu *payload*, o recurso solicitado é liberado para o usuário.

O protocolo de encerramento de sessão (*Single Logout Profile*) define a sequência de trocas de mensagem que determinam a saída organizada de um cliente de sua sessão de computação, que pode envolver vários SPs acessados pelo cliente durante a sessão de computação. A Tabela 4 mostra os passos de um *Single Logout* no *SAML*.

Tabela 4 – Protocolo de *Single Logout* no SAML

Protocolo de Logout no SAML	
Passo	***Descrição***
1	O cliente (participante da sessão) emite um **<LogoutRequest>** para o Provedor de Identidades;
2	O Provedor de Identidades determina quais são os participantes da sessão de computação do cliente e envia a mensagem ***<LogoutRequest>*** para todos estes participantes;
3	Os participantes respondem para o Provedor de Identidades com a mensagem **<LogoutResponse>**, informando que encerram a sessão;
4	Provedor de Identidades então envia para o cliente uma mensagem **<LogoutResponse>** informando que a sessão foi encerrada.

Suporte a federação no SAML

A especificação *SAML* 2.0 provê suporte para a implementação de uma federação de Provedores de Identidades. A utilização de identidades federadas permite que o gerenciamento de identidades atravesse as fronteiras organizacionais, de forma que estas organizações compartilhem dados sobre identidades envolvendo todos os domínios dos *IdP*s associados na federação. Estas transposições de identidades entre domínios podem não significar quebras de privacidade. A especificação *SAML* também define suporte para pseudônimos, que são identificadores dinâmicos que não remetem exatamente a uma identidade (sujeito) específica.

b) *SHIBBOLETH*

Shibboleth (MORGAN; SCAVO, 2005) é um projeto *Open Source* que fornece uma solução *SSO* e acesso a serviços *web* de forma segura e usando identidades federadas. O *Shibboleth* é baseado em *SAML*, utilizando os protocolos e asserções desta especificação tanto para o processo de autenticação e como para a autorização. Atualmente, esta infraestrutura é amplamente adotada como suporte para identidades federadas em diversos setores, tais como universidades, organizações de pesquisa e órgãos governamentais.

Esta infraestrutura de autenticação está fundamentada sobre padrões abertos como o *XML* e o *SAML* e provê uma forma fácil para que aplicações *web* usufruam das facilidades disponibilizadas pelo modelo de identidades federadas, como o conceito de autenticação única, a troca segura de atributos e o *Single Logout* para serviços e usuários que pertençam à federação. O *Shibboleth* tem como ênfase a privacidade dos

atributos dos usuários, sendo que a liberação desses atributos para os provedores de serviços está condicionada à política de privacidade da instituição de origem do usuário e também às preferências pessoais do usuário.

Dentro de um domínio *Shibboleth*, existem dois papéis: provedor de identidades (*Identity Provider – IdP*) e provedor de serviços (*Service Provider – SP*). O primeiro é responsável por autenticar seus usuários, antes que estes possam usufruir dos serviços oferecidos pelo segundo. O ponto comum entre estes papéis é que ambos devem implementar toda a pilha de *software* fornecida pelo projeto *Shibboleth*, permitindo assim o transporte das credenciais dos usuários do provedor de identidades até o provedor de serviços.

No *Shibboleth*, o processo de autenticação sempre é executado na instituição de origem do usuário, através de seu provedor de identidades, fazendo uso dos mecanismos de autenticação presentes nessa instituição. A Tabela 5 mostra os passos da autenticação em uma federação *Shibboleth*.

Em domínios de federação *Shibboleth*, o *SP* é o responsável por aplicar o controle de acesso, considerando sempre os atributos do usuário fornecidos na requisição de acesso. O provedor de serviços pode requisitar outros atributos do usuário junto ao provedor de identidades do usuário para usar no mecanismo de controle de acesso. O *Shibboleth* define uma forma padronizada para troca de atributos, através de asserções *SAML*.

Tabela 5 – Protocolo de Autenticação *Shibboleth*

Protocolo de Autenticação no Shibboleth	
Passo	***Descrição***
1	O usuário, através de seu *browser*, requisita acesso a uma aplicação web;
2	A aplicação verifica que o usuário ainda não efetuou o processo de autenticação;
3	(Opcional) Para que o usuário se autentique, este é redirecionado (*HTTP Redirect*) para o serviço de descoberta *Where Are you From* (*WAYF*);
4	O usuário então seleciona em uma lista de provedores de identidades aquele *IdP* que contém suas credenciais e pode autenticá-lo;
5	O usuário é redirecionado pelo serviço de descoberta para o provedor de identidades selecionado;
6	O usuário então inicia o processo de autenticação, através da inserção de suas credenciais (usuário/senha, certificado, *LDAP*, etc);
7	Caso o usuário insira corretamente suas credenciais o provedor de identidades cria uma asserção de autenticação, que persiste durante toda a sessão de computação do usuário;
8	O usuário é então redirecionado para o serviço (*HTTP Redirect*) assim como a asserção de autenticação (asserção SAML).

c) *WS-FEDERATION*

A especificação *WS-Federation* (OASIS, 2009) é um padrão OASIS que define mecanismos de identidade federadas baseados nos padrões *WS-Security*, *WS-Trust* e *WS-Policy*. Esses padrões já definem as

bases para o gerenciamento federado de identidades, porém a *WS-Federation* introduz extensões que determinam como combinar conceitos e protocolo dos padrões (anteriores) da família *WS-* ∗. Esta combinação dos padrões citados forma um conjunto de funcionalidades bastante interessantes se considerarmos o conceito de federação e domínios de segurança.

Há alguma sobreposição entre as funcionalidades da *WS-Federation* e da tecnologia *SAML*. Ambas soluções permitem o gerenciamento de identidades federadas e provêm um conjunto de funcionalidades similares. Assim como as especificações *SAML*, a *WS-Federation* suporta autenticação única, encerramento único de sessão, compartilhamento de atributos e pseudônimos. A principal diferença é que a *WS-Federation* está baseada no modelo de confiança da *WS-Trust* e não o baseado em asserções *SAML*.

Uma vez que *WS-Federation* é uma extensão de *WS-Trust*, este deve seguir o mesmo protocolo para a administração e troca *tokens* de segurança, utilizando um serviço denominado *Security Token Service* (*STS*). Na especificação *WS-Federation* o *STS* acumula os papéis de provedor de identidades e de emissor de *tokens* de segurança, sendo conhecido como *IdP*/*STS*. Assim como na especificação *SAML*, são definidos nas especificações *WS-Federation* protocolos de *Single Sign-On* e *Single-Logout*.

No contexto de Serviços *Web*, o objetivo destas especificações é permitir que identidades e atributos sejam negociados entre o *IdP/STS* e os provedores de serviços, sem a necessidade de intervenção do usuário. As federações podem existir dentro de uma organização ou estar além de seus domínios administrativos. Diferentes cenários de federação são apresentados na especificação *WS-Federation*, os quais são apresentados pela Figura 5.

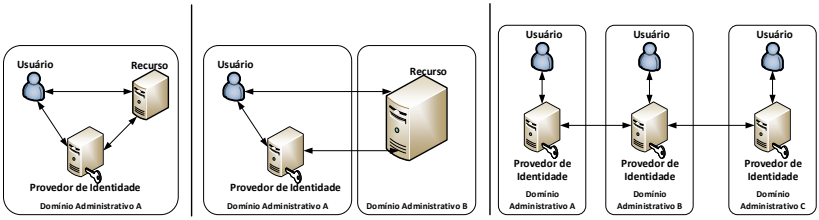


Figura 5 – Modelos de Federação *WS-Federation*

No primeiro cenário, tanto o usuário quanto o recurso a ser acessado pertencem ao mesmo domínio administrativo, onde as relações de confiança entre o recurso e o cliente são mediadas por um provedor de

identidade interno da organização. Já no segundo cenário, um contexto de segurança entre domínios administrativos distintos é estabelecido, de forma que um usuário (Domínio Administrativo A) utilize recursos em outro domínio (Domínio Administrativo B). Para isso, o provedor de identidade e o recurso, pertencentes a domínios distintos, devem manter uma relação de confiança. Para que o recurso seja liberado para um usuário do domínio do *IdP*, é necessário que este *IdP* emita uma asserção de autenticação de seu usuário para que o SP deixe o mesmo acessar seus serviços. Já o terceiro cenário representa relações de confiança entre os provedores de identidades dos diversos domínios administrativos.

A gramática utilizada pelo *WS-Federation* é o XML, e as trocas de mensagens se baseiam nos *protocolos SOAP***,** de forma que siga os padrões das outras especificações *WS-**. Apesar da gramática da especificação *WS-Federation* ser similar ao *SAML*, não há compatibilidade entre as duas especificações**.**

d) PROJETO *LIBERTY ALLIANCE*

Liberty Alliance Project (LIBERTY, 2003) é um projeto formado por um consórcio de empresas de diferentes áreas com o objetivo de estabelecer padrões e boas práticas para o gerenciamento de identidades federadas. Dentre os objetivos do projeto estão preocupações relacionadas à segurança e privacidade das informações dos usuários, utilização de padrões abertos, de forma que estes provedores de identidades interoperem, permitindo que o processo de *Single Sign-On* seja realizado dentre diversos provedores, utilizando protocolos e especificações que sejam aceitas e compatíveis com qualquer tipo de dispositivo ou tecnologia.

O projeto é composto pelos módulos *Identity Federation Framework* (*ID-FF*), *Identity Web Services Framework* (*ID-WSF*) e *Identity Services Interface Specifications* (*ID-SIS*) que são descritos na sequência:

- *Liberty Identity Federation Framework (ID-FF)*: define o gerenciamento de identidades federadas através de diversos domínios administrativos, caracterizando o que é chamado de "círculo de confiança". Este círculo de confiança é formado por organizações através de relações de confiança e acordos comerciais. Nestes círculos de confiança cada organização pode assumir papéis como provedores de identidades (*IdP*), como provedores de serviços (SP) ou como ambos.

- *Liberty Identity Web Services Framework (ID-WSF)*: é um dos módulos fundamentais da *Liberty Alliance*, que define uma estrutura para a criação e descoberta de Provedores de Identidades (*IdP*s). As especificações deste módulo definem perfis de segurança e de integração de protocolos (como o SOAP e o *HTTP*) para interação com o cliente. Portanto, assim como nos demais módulos, neste é também recomendada a utilização dos padrões e especificações já existentes e amplamente aceitas.
- *Liberty Identity Services Interfaces Specifications (ID-SIS)*: Neste módulo são definidas as especificações de interface para a interação do *framework Liberty Alliance* com os serviços que irão prover identidades (*IdP*s). As especificações são descritas de tal forma que as organizações possam rapidamente e facilmente estender os provedores de identidade já existentes ou criar serviços adicionais sobre o *framework ID-WSF*.

O foco da infraestrutura *Liberty Alliance* não é a de criar novos padrões ou propor novas especificações, mais sim adotar padrões já utilizados.

e) *OPENID*

O *OpenID* é um *framework* para gerenciamento de identidades que segue a abordagem centralizada e usa protocolos e padrões conhecidos, como *HTTP* (R. FIELDING et al., 1999), HMAC(H. KRAWCZYK; BELLARE; CANETTI, 1997), "*Diffie-Hellman Key Agreement Method*"(E. RESCORLA, 1999), SHA1 (D. EASTLAKE; JONES, 2001) e outros. Neste *framework*, o protocolo de identificação inicia com um usuário se apresentando a um provedor de serviços (chamado de *Relying Party* nas especificações *OpenID*) através de um identificador *OpenID*. Este identificador *OpenID* é usado em vários provedores de serviço (*RP - Relying Parties*) que confiam no *IdP OpenID* gerador deste identificador.

Esta confiança é necessária porque os provedores de serviços (*RPs*) na sequência, devem delegar a autenticação do usuário ao provedor de identidades (*OpenID Identity Provider*) indicado pelo identificador fornecido pelo processo no *login*. Estes identificadores de usuários podem ser *URLs* ou, ainda, documentos *XRI* (*Extensible Resource Identifier*), que fornecem informações adicionais, como *URLs* de provedores de identidades onde as credenciais dos usuários possam ser aceitas.

f) *OpenID Connect*

OpenID Connect (OPENID, 2014) é uma atualização das especificações *OpenID* 2.0. Assim como a versão anterior, esta última é composta por um conjunto simples de especificações, porém nesta versão todos os seus protocolos são acessíveis através de uma API *REST*. Basicamente, *OpenID Connect* é uma camada de gerenciamento de identidades implementada no sobre o protocolo OAuth 2.0 (MICROSOFT, 2012), onde controles de autorização também podem ser aplicados.

Assim como na especificação *OpenID* 2.0, os elementos principais do *framework OpenID* Connect são o cliente, o provedor de serviços e o provedor de identidades. O cliente é o elemento que deseja acessar recursos protegidos em um provedor de serviços (*Resource Provider – RP* nas especificações *OpenID*). O Provedor de Identidades é a entidade que armazena as credenciais do usuário e é capaz de gerar asserções de autenticação, que são aceitas como forma de liberar recursos para o usuário pelo Provedor de Serviços. Tanto o *OpenID* como o *OpenID Connect* são desenvolvidos em mais detalhes em outros capítulos deste texto porque serviram de base para nossos trabalhos.

2.4 COMPUTAÇÃO EM NUVEM (*CLOUD COMPUTING*)

O *National Institute of Standards and Technology* (NIST) define a computação em nuvem como um modelo ubíquo (disponível para conexão de qualquer lugar, em qualquer momento) de acesso sob demanda através de rede a um *pool* compartilhado de recursos computacionais configuráveis (como, serviços de armazenamento e outras infraestruturas, de plataforma, de aplicações e de redes) que podem ser rapidamente provisionados e liberados com um esforço de gerenciamento mínimo ou interação com o provedor destes recursos (MELL; GRANCE, 2011).

Usuários da computação em nuvem utilizam recursos computacionais alocados em *data centers* de posse de grandes provedores através de um modelo denominado "*pay-as-you-go*". Neste modelo, os usuários pagam por recursos de acordo com sua utilização, geralmente contabilizada por horas de utilização.

O modelo de computação em nuvem vem se consolidando nos últimos anos e vem recebendo, ainda que de forma gradual, a adesão cada vez maior de empresas a este estilo de computação. Esta adoção da computação em nuvem implica no fato das empresas migrarem suas aplicações computacionais, inicialmente executadas em recursos locais

das empresas, para grandes provedores que disponibilizam seus *data centers* para estas empresas (os seus usuários externos). Os ganhos da computação em nuvem estão relacionados a que empresas passam a não manter grandes infraestruturas computacionais para atender suas necessidades de processamento, mas sim de utilizarem, por preços geralmente convidativos, recursos destes grandes provedores de *cloud*.

Apesar de ainda não existirem padrões referentes a *clouds* e suas tecnologias, algumas características e tendências estão se delineando nestes últimos anos referentes a este estilo de computação. Nas subseções seguintes são descritos os modelos de computação em nuvem atualmente adotados, bem como suas características.

2.4.1 Características dos Serviços de *Cloud*

De acordo com o NIST (MELL; GRANCE, 2011), a computação em nuvem é baseada em cinco (5) características essenciais:

a) *Serviços sob Demanda* (*On-demand self-service*): a ideia, neste caso, é de que um usuário poderá ter provido ou devolvido recursos de acordo com suas necessidades no momento e de forma automática, sem necessidade de interação humana.

b) *Acesso amplo via rede (Broad network access*): os recursos devem estar disponíveis via rede e devem poder ser acessados pelos mais diversos tipos de dispositivos, tais como computadores, telefones *smartphones*, *tablets*.

c) *Pooling de recursos* (*Resource Pooling*): os recursos computacionais oferecidos pelos provedores são agrupados de forma a atender a um grande número de usuários (clientes de serviços de *cloud*). Este modelo de compartilhamento de recursos entre múltiplos clientes é conhecido na literatura de *clouds* como modelo "*multi-tenant*". Os recursos físicos de *data centers* (computadores, redes, etc.) são divididos em recursos virtuais (máquinas virtuais, redes virtuais), atribuídos e realocados de forma dinâmica e independente de localização, sem que o cliente tenha conhecimento desta flexibilidade.

d) *Elasticidade* (*Rapid Elasticity*): o usuário não deve ter como preocupação a variação de carga de suas aplicações, uma vez que recursos são alocados sempre que necessários. Para o usuário, fica sempre a noção do provedor de *cloud* ter recursos infinitos.

e) Serviços mensurados (*Measured Service*)***:*** apesar da dinâmica na alocação de recursos, para o usuário deve ficar transparente o quanto de recursos estão sendo utilizados em suas aplicações e a carga que está sendo imposta a estes recursos. A transparência

nestas medições permite ao usuário ter a noção sobre os custos que este estará sujeito na utilização dos recursos.

2.4.2 Níveis de Serviços em *Clouds*

Os serviços de *clouds*, normalmente, são divididos segundo o nível de recursos que fornecem aos seus clientes. Os mais usuais encontrados na literatura são (MELL; GRANCE, 2011): Infraestrutura como Serviço (*Infrastructure as a Service - IaaS*); Plataforma como Serviço (*Plataform as a Service - PaaS*) e *Software* como um Serviço (*Software as a Service - SaaS*). Estes três níveis de serviço são ilustrados na Figura 6.

No nível de infraestrutura (*IaaS*) são oferecidos aos clientes serviços de processamento (máquinas virtuais), armazenamento (discos virtuais), redes e outros recursos básicos de computação. A partir destes recursos básicos, o cliente deve implementar e executar os *softwares* de suas aplicações. Neste modelo, o usuário também tem a liberdade de incluir os sistemas operacionais de sua escolha e plataformas especiais para executar seus aplicativos. Recursos computacionais de mais baixo nível, que mantêm estes serviços de infraestrutura, não podem ser visíveis ou controlados pelos clientes. Neste nível os usuários dos serviços de *cloud* possuem o controle sobre as pilhas de *software* por eles introduzidas sobre os recursos de infraestrutura disponibilizados pelo seu provedor de *cloud*.

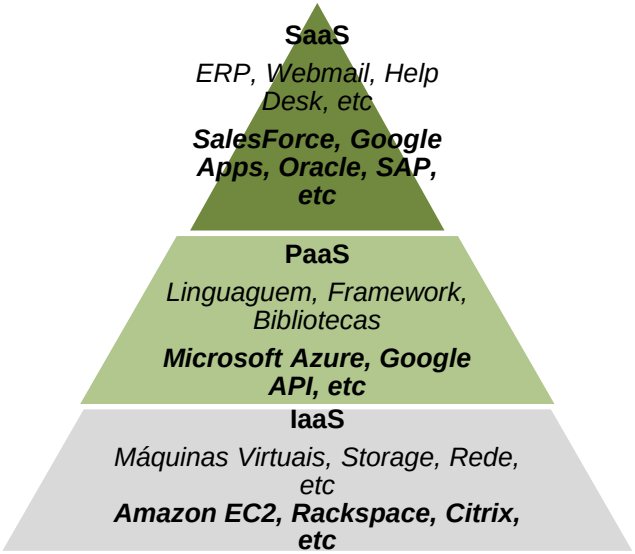


Figura 6 – Níveis de Serviços em *Clouds*

No nível de plataforma (*PaaS*), são oferecidos para o cliente recursos de plataforma para que este possa desenvolver e implantar suas aplicações. Uma plataforma inclui linguagens de programação, bibliotecas, serviços e ferramentas para o suporte ao desenvolvimento de aplicações. Neste nível, o cliente não gerencia e nem controla recursos de nível de infraestrutura onde estão seus recursos de plataforma. Ou seja, os dispositivos de rede, sistemas operacionais ou recursos de armazenamento não estão disponíveis diretamente para o cliente. Os controles ficam restritos ao ambiente de desenvolvimento e às aplicações desenvolvidas sobre os recursos de plataforma.

No nível de Software (*SaaS*) é oferecido para o cliente a utilização de aplicações específicas sendo executados em recursos de *clouds*. Estas aplicações são disponibilizadas pela Internet, e são acessíveis a partir de vários dispositivos clientes (computadores, *smartphones*, *tablets*). O cliente não gerencia nem controla nenhum recurso da *cloud*, ficando restrito às configurações da aplicação que este contratou (usuários, parâmetros de funcionamento, dentre outros).

2.4.3 Modelos de *Clouds* (Modelos de implantação)

A classificação normalmente aceita divide os modelos de implantação de *clouds* em quatro classes (MELL; GRANCE, 2011):

1. *Nuvem privada*: corresponde a infraestruturas de nuvem que fornecem recursos exclusivos para uma única organização (ou domínio administrativo). A infraestrutura é operada pela própria organização, ou terceirizada ou ainda a combinação dos dois primeiros.
2. *Nuvem comunitária*: infraestrutura de nuvem que fornece recursos para uso exclusivo de consumidores de uma comunidade específica, com suas aplicações e objetivos comuns, dentro de uma perspectiva de integração e compartilhamento. Assim como nuvens privadas, as comunitárias também podem ser gerenciadas por terceiros (uma ou mais empresas).
3. *Nuvem pública:* corresponde ao tipo de nuvem mais comum atualmente. Esta infraestrutura de nuvem é fornecida para uso aberto ao público em geral. Pode ser gerenciada e operada por uma empresa, universidade, organização governamental ou alguma combinação desses.
4. *Nuvem Híbrida*: são infraestruturas de nuvem formadas por uma composição de nuvens dos tipos citados acima (privada, comunitária ou pública) que permanecem como entidades únicas. As *clouds*

híbridas são criadas para formar infraestruturas maiores, preservando as características dos tipos combinados e fornecendo uma maior gama de serviços.

2.4.4 Federação de Recursos de *Clouds*

Na literatura de *clouds*, o conceito de federação vem sendo discutido com alguma frequência. Porém, não existem conceitos que definam especificamente uma *federação de clouds*, mas sim experiências significativas na literatura que indicam o interesse nesta linha de evolução.

A noção de federação está fortemente associada a relações de confiança. As redes de confiança que se formam sob o chapéu de uma "federação" definem relações próprias entre provedores de algum tipo de serviço, no sentido da colaboração dos mesmos no atendimento de requisições de seus clientes.

No contexto de *clouds*, a utilização de federações permite que provedores de serviços (aplicações web, *SaaS* em geral) e de recursos (*IaaS* ou *PaaS*) colaborem de forma a compartilhar seus serviços ou recursos. Os tipos de compartilhamento são os mais refinados possíveis (máquinas virtuais, armazenamento, redes virtuais, plataformas, software, etc.)(KURZE et al., 2011). Este compartilhamento visa atender diferentes situações, tais como a falta de recursos em um provedor para atender a grandes demandas ou suprir um serviço originalmente não oferecido pelo provedor, distribuição de uma aplicação de usuário em diferentes provedores, dentre outras.

2.4.5 Cloud Brokers

Uma infraestrutura de nuvem, em seus diversos níveis (*IaaS*, *PaaS*, *SaaS*), implica certa complexidade no que se diz respeito à provisão de recursos de qualquer natureza para seus usuários. A forma com que os recursos são gerenciados e adquiridos não deve ser preocupação do usuário, mesmo que usando recursos em uma federação de recursos. Para isto, as propostas de federação de recursos (federação de provedores de *cloud*) introduzem, de maneira geral, a figura de um *cloud broker* (também existente em *clouds* não federadas) que é uma entidade proposta para, em nome do cliente, gerenciar, alocar e analisar o desempenho de recursos que suportam as demandas do usuário.

Os recursos gerenciados pelos *cloud brokers* podem ser locais, de um mesmo provedor, ou recursos de uma federação, em caso de gestão em *clouds* federadas. Na Figura 7 é apresentada uma visão geral de um

cloud broker e suas relações com seus diversos usuários, *modelos de implantação* e serviços (JULA; SUNDARARAJAN; OTHMAN, 2014).

De maneira geral, um *cloud broker* deve fornecer serviços em três categorias (MELL; GRANCE, 2011):

1. *Intermediação de serviços*: um *broker* deve intermediar todas as ações entre o usuário e o provedor, tais como geração de relatórios de desempenho, gerenciamento de identidades, implantação automatizada de serviços, dentre outros.
2. *Agregação de serviços*: combina e integra múltiplos recursos em um ou mais serviços, com o objetivo de oferecer uma funcionalidade não existente em um provedor específico.
3. *Arbitragem de serviços*: esta funcionalidade é similar à agregação de serviços. Os provedores são agregados tomando como base algum critério específico (preço, qualidade de serviço). Por exemplo, um provedor compra um serviço (armazenamento de dados, banda de internet) de um provedor e os revende para seus usuários.

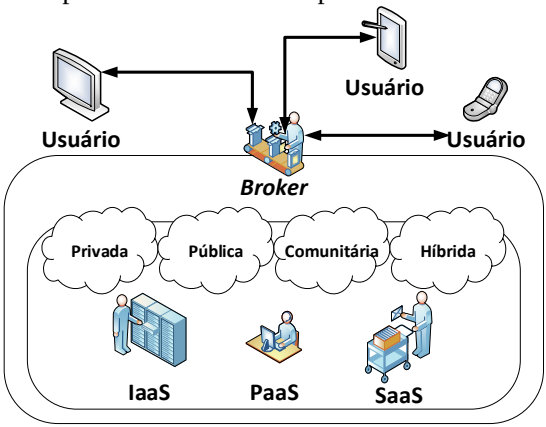


Figura 7 – *Cloud Broker*

2.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os conceitos básicos de Segurança Computacional, Gerenciamento de Identidades e Computação em Nuvem. Estes conceitos são usados nos demais capítulos, e, quando necessário, serão revisados ou revisitados para uma melhor compreensão.

Na terceira parte do capítulo, nos concentramos nos aspectos referentes à computação em nuvem e seus modelos. Apresentamos conceitos referentes aos diferentes modelos de serviço presentes nos provedores de computação em nuvem, assim como nas características essenciais deste modelo. Ainda sobre computação em nuvem, descrevemos o conceito de federação de provedores de *cloud*, no qual os

trabalhos descritos neste documento usam como contexto principal. Por fim, introduzimos o conceito de *cloud brokers*, entidade essencial em *clouds* e principalmente na formação de federação de *clouds*.

Nos capítulos subsequentes exploraremos as bibliografias relacionadas e apresentaremos a nossa abordagem para federações de provedores de *cloud* e os trabalhos desenvolvidos para serviços de autenticação tolerantes a intrusões.

3 FEDERAÇÃO DE *CLOUDS*

Neste capítulo descrevemos um modelo de Federação de *Cloud*. A organização estrutural apresentada serviu inicialmente de contexto para os nossos trabalhos de autenticação e autorização em federação de *clouds*. Definimos este modelo como baseado em *broker*. Além do modelo, apresentamos os resultados obtidos através de simulações do modelo estrutural, que serviram para analisar o seu comportamento em diferentes situações de provimento de recursos pela federação de *clouds* diante de demandas de usuários. Por fim, apresentamos a literatura relacionada a este modelo e as conclusões da seção e do modelo.

3.1 MOTIVAÇÃO

Nos últimos anos, a computação em nuvem tem sido foco de grande parte dos esforços de pesquisa da comunidade de sistemas distribuídos. Muitas das noções pré-estabelecidas de sistemas e de aplicações distribuídas vêm sendo revistas diante dos requisitos postos por estes novos e complexos ambientes de processamento distribuído.

As grandes demandas que esses sistemas devem suportar têm feito dos mesmos foco de experimentações e de busca de novos conceitos, objetivando manter bons níveis de aproveitamento de recursos com alta qualidade de serviço. O conceito de *federação*, por exemplo, tem sido explorado na literatura de *clouds*, visando a cooperação entre diferentes provedores. A integração em federações permite que provedores de *clouds* colaborem de forma a compartilhar recursos no atendimento a demandas de usuários. Os tipos de compartilhamento podem ocorrer em diversos níveis de recursos (máquinas virtuais, armazenamento, redes virtuais, plataformas, software, etc.) (KURZE et al., 2011).

A cooperação entre *clouds* pode atender diferentes situações, tais como: a falta de recursos em um provedor para suportar grandes demandas; suprir serviços ou recursos originalmente não oferecidos por um provedor, a distribuição de uma aplicação, definida pelo próprio usuário, entre diferentes provedores; e a colaboração entre pequenos provedores de *clouds*, dentre outros.

O interesse atual na investigação do uso da noção de federação de *clouds* é, portanto, justificável. Adicionalmente, o agrupamento de provedores de *clouds*, formando redes de confiança que atendem um vasto número de usuários, resulta também em grandes desafios na autenticação e autorização destes usuários junto aos provedores da federação. Para contornar possíveis dificuldades no trato de aspectos de segurança, modelos de gerenciamento de identidades se tornam importantes para estes grandes sistemas.

O foco principal deste capítulo é a apresentação de nossa proposta de federação de *clouds*, onde identificamos as principais entidades participantes que suportam o conceito de federação. Descrevemos também os protocolos usados na busca e aquisição de recursos na federação de *clouds*, em atendimento às demandas de usuários. Estes protocolos comportam também mecanismos que visam atender requisitos de segurança, como os controles de autenticação e de autorização na federação de *clouds*. O conceito de gerenciamento de identidades é a base para estes controles de segurança, bem como para a própria federalização de provedores e usuários em nosso modelo.

Os serviços de federação são fornecidos nesta proposta a partir de uma infraestrutura de serviços e de um provedor de identidades. As diversas entidades (provedores e usuários) da federação de *clouds* confiam nos serviços desta infraestrutura e nas autenticações deste provedor de identidades, o que permite que provedores de *clouds*, por si só, não se preocupem em manter extensas bases de dados com identidades e atributos de usuários.

3.2 FEDERANDO PROVEDORES DE *CLOUD*

Em (GROZEV; BUYYA, 2012), o conceito de federação de *clouds* é apresentado como um conjunto de provedores que, voluntariamente, interconectam suas infraestruturas com o objetivo de compartilhar seus recursos. Porém, além do termo federação, outros termos são usados para nomear a cooperação de provedores no compartilhamento de recursos. Entre outros, encontramos na literatura termos como: *Inter-cloud* (GROZEV; BUYYA, 2012), *Multi-cloud* (GROZEV; BUYYA, 2012) e *Cross-Cloud* (CELESTI et al., 2010b). As diferenças na nomenclatura e mesmo que muitas destas propostas na formação de redes de confiança entre provedores de *cloud* tenham diferentes focos, isto não impede que o objetivo final seja sempre a cooperação entre provedores de *clouds* para melhor atender a demanda e *SLA*s de seus usuários.

3.2.1 Proposta de Federação

O modelo introduzido nesta seção foi desenvolvido para que usuários, diante da necessidade de recursos computacionais, emitam seus *SLAs* para que uma entidade de suporte, baseada nas informações de demanda, busque por recursos na federação de *clouds* que atendam a requisição do usuário. Em nossas proposições, os usuários não se preocupam em como ou onde os recursos estão disponíveis para seu uso. A localização dos recursos é sempre controlada por uma infraestrutura global de serviços que suporta a federação de *clouds*. Esta ideia de transparência de localização já está presente em vários dos trabalhos citados nesta seção.

Uma visão geral de nossa proposta de federação é ilustrada pela Figura 8, onde as diferentes entidades do modelo são representadas. A primeira destas corresponde a uma *stub/API* denominada de *Cloud User*, disponível no ambiente do usuário. Este componente permite aos usuários interagir com a federação, enviando *SLAs* e requisições de recursos. Para o usuário, a federação oferece certa facilidade para a aquisição destes recursos, uma vez que este somente precisa definir contratos de utilização de recursos.

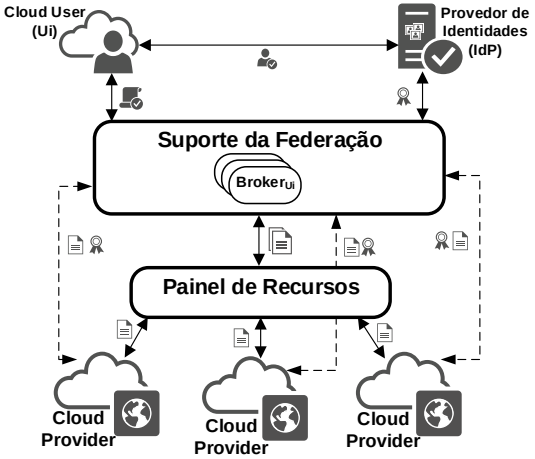


Figura 8 – Visão geral da proposta de federação

As interações do usuário com a federação em nosso modelo são realizadas por meio de uma infraestrutura chamada de Suporte da Federação (*SF*), que é responsável por receber as requisições de recursos dos usuários e instanciar *brokers* para negociar e controlar diretamente o *SLA* de seu usuário na federação de *clouds*. Para cada contrato de usuário (*SLA*) em nosso modelo, um *broker* é instanciado e permanece ativo até o fim deste contrato. Cada *broker* atua em nome do seu usuário na aquisição dos recursos, entregando *tokens* (asserções) de autenticação aos provedores de *cloud* de forma a comprovar estes requisitantes como usuários válidos na federação.

O modelo pressupõe o envolvimento de diversos provedores de *clouds* (*Cloud Providers*), que são os fornecedores de recursos da federação. Estes provedores buscam preencher as demandas de *brokers*, feitas em nome de seus usuários. Cada provedor de *cloud* atua na federação como uma entidade única, oferecendo diferentes tipos de serviço e zelando única e exclusivamente pelos seus próprios recursos. Estes provedores interagem com os *brokers* no momento da busca e

aquisição dos recursos. Após este processo, os recursos considerados são mantidos disponíveis aos usuários envolvidos. Os provedores de *cloud* mantêm seus próprios controles de segurança (autorização), de forma que somente ofereçam recursos para usuários que tenham a posse de um *token* de autenticação. Depois de feitos disponíveis a um usuário, no cumprimento de um *SLA*, os recursos passam a ter os controles de acesso definidos pelo próprio usuário.

Nas requisições de recursos, as interações entre *brokers* e provedores de *cloud* no nosso modelo são concretizadas através do componente Painel de Recursos (*PR*), que se comporta de forma semelhante a um serviço de eventos. Para receberem pedidos vindos de *brokers*, provedores devem se registrar no *PR* como receptores de notificações adequadas a seus recursos disponíveis. Os *brokers* também devem se registrar em suas inicializações como emissores de notificações (baseados nas demandas de seus usuários). No Painel de Recursos são mantidas listas de provedores ativos e registrados. Listas (associadas) agrupam provedores por tipo de recursos que estes tornam disponíveis na federação. Determinados provedores de *cloud* consomem notificações de certo tipo, porque fazem parte da lista associada do tipo citado.

No recebimento de demandas, o Painel verifica quais são os recursos solicitados e notifica diretamente os provedores possuidores destes recursos, fazendo uso das listas associadas que mantém. Os provedores então respondem diretamente ao *broker* correspondente com as suas disponibilidades dos recursos solicitados.

Provedores de Identidades (*IdPs*) são introduzidos no modelo com funções de manter, gerenciar e autenticar usuários e seus atributos na federação. Cada usuário, no acesso ao Suporte da Federação (*SF*), deve inicialmente se autenticar junto a um destes *IdPs*. Autenticidades verificadas geram *tokens* (ou asserções) de autenticação que devem garantir aos usuários validados a busca e o uso de recursos nos diversos provedores de *cloud* da federação. Os provedores de *cloud* confiam nas autenticações de usuários dos *IdPs*. Recursos só são liberados para usuários previamente autenticados na federação.

3.2.2 Componentes e Protocolos

Nesta seção são apresentados os algoritmos que descrevem como são feitas as trocas na busca e alocação de recursos entre as entidades do modelo. Estas buscas e acessos aos recursos são acompanhados por controles de autenticação e de autorização dos usuários da federação.

a) *Cloud User*

O Algoritmo 1 apresenta o processo de solicitação de recursos na federação, onde o usuário inicialmente gera seu *SLA* (linhas 13 – 17), especificando os parâmetros de contrato (preço, *uptime* mínimo, qualidade mínima de serviço, dentre outros).

Os tipos de cada recurso desejado são selecionados (linhas 19 – 22, Algoritmo 1) na sequência. A requisição $fedReq_i$ é montada (linha 25) e armazenada localmente no *stub Cloud User* (linha 26) para ser enviada após ao Suporte da Federação (linhas 27). Uma vez enviada a requisição ao *SF*, em um primeiro momento, o usuário deve receber um pedido de autenticação ($authCReq_i$) que vem do provedor de identidades, como consequência de sua requisição enviada ao Suporte da Federação. O usuário então responde a este *IdP* com suas credenciais (usuário/senha, certificado, etc) através da mensagem $authCred_i$ (linhas 28–30). Após este passo, o usuário aguarda pelos ponteiros dos recursos, que são enviados pelo *SF*, via o *broker* alocado para o seu *SLA*, através de uma mensagem $fedRep_i$ (linhas 31 e 32).

Todas as mensagens trocadas entre as entidades do sistema são assinadas de modo a garantir a autenticidade e a integridade das mensagens. Ou seja, as entidades do modelo só trocam informações com outras entidades credenciadas e devidamente validadas. Por exemplo, com a mensagem $fedReq_i$ (linha 25) é enviada a assinatura $sign_{ui}$ e o certificado de chave pública do assinante ($cert_{ui}$) para validação posterior desta mensagem[4].

b) *SF* (Suporte de Federação)

A entidade identificada como Suporte de Federação (*SF*) é responsável por receber requisições e instanciar *brokers* para atender as solicitações de usuários. O algoritmo 2 descreve seu comportamento nas interações com os *Cloud Users*. Neste algoritmo, ao receber uma requisição $fedReq_i$ de usuário (linha 3), o *SF* armazena esta requisição e recupera algumas informações importantes, como por exemplo, o identificador do usuário (linhas 4 – 6, algoritmo2).

[4] Os certificados usados na nossa proposta não foram pensados como parte de *PKI´*s envolvendo *CA´*s (Autoridades Certificadoras) oficiais. Estes certificados podem ser emitidos por entidades como um comitê gestor da federação ou qualquer entidade entendida como confiável pela federação.

Algoritmo 1 – *Cloud User*

Algoritmo 1. *Cloud User* U_i

```
Structures
1:    Resource {                                   //Define os recursos que deseja requisitar
2:      Type;
3:      Amount;}
4:  Type:
5:    VmTypeA = {1ghz cpu, 1gb mem, ...}         //Template do caso de Recursos VM
6:    VmTypeB = {2 ghz cpu, 2 gb mem, ...}
7:  Init:
8:    n_u ← 0;                                   //Identificação da requisição
9:    fedReqList[] ← Ø;                          //Lista de Pedidos já efetuados
10:   res[] ← Ø;                                 //Lista de Recursos sendo requisitados

11:   upon ResourceRequest do                    //Processo de requisição de recursos
12:     //Cria Requisito de SLA
13:     price ← ≥ X;                             //Define preço desejado
14:     uptime ← ≤ 99%;                          //Define uptime desejado
15:     QoS ← ≥Y;                                //Define outros parâmetros de QoS
16:     ....
17:       sla_i ← <SLA, price, uptime, QoS, ...> //Gera o SLA do pedido
18:       //Cria requisitos do pedido
19:       res ← res ∪ Resource(VmTypeA, 2);      //Adiciona VmTypeA
20:       res ← res ∪ Resource(VmTypeB, 3);      //Adiciona VmTypeB
21:       …
22:       resReq_i ← <RESREQ, res>;              //Mensagem contendo recursos
23:       n_u++; //Numeração da mensagem
24:       id_req ← <IDREQ, id_Ui, n_u>;          //Identificação da mensagem
25:       fedReq_i ← <FEDREQ, id_Ui, id_req, sla_i, resReq_i, sign_ui, cert_ui>  //Gera e assina a requisição
26:       fedReqList ← fedReqList ∪ fedReq_i;    //Adiciona a requisição
27:       send fedReq_i to SF                    //Envia a requisitção para o SF

28:   upon Receive authCReq_i from IdP do        //Processo de pedido de autenticação
29:     authCred_i ←<AUTHCRED_Ui, id_Ui, id_req, credentials, sign_ui, cert_ui>;  //Faz o envio das credenciais
30:     send authCred_i to IdP                   //Envia para o provedor de identidades

31:   upon Receive fedRep_i from B_i do          //Ao receber resposta dos Brokers
32:       resourceRequested ← fedRep_i.fedAlRes; //Recursos alocados para uso
```

De posse do identificador id_{ui} do usuário, o *SF* executa um procedimento de descoberta (MORGAN; SCAVO, 2005)(MILLER, 2006) (linha 7) que visa encontrar o Provedor de Identidades que armazenas as credencias e atributos do usuário. Executado este protocolo de descoberta, o Suporte de Federação (*SF*) envia então uma requisição de autenticação ($authReq_i$) ao *IdP* correspondente (linhas 8 e 9, algoritmo 2).

Após o envio da requisição de autenticação, o Suporte de Federação aguarda um *token* de autenticação ($authToken_i$) vindo do *IdP* que confirme o emissor da requisição como usuário válido no sistema. Uma vez recebido este *token*, o *SF* faz as verificações necessárias da assinatura do *token* e, se sua validade for confirmada, um *broker* é então instanciado para atender as solicitações do usuário (linhas 10 -14, algoritmo 2).

Algoritmo 2 – Suporte da Federação

Algoritmo 2. *Suporte da Federação F*

```
1.  Init:
2.    fedReqList ← Ø;                                   //Lista de pedidos de recursos
3.  upon Receive fedReq_i from U_i do                   //Ao receber um pedido de usuário
4.    fedReqList ← fedReqList ∪ fedReq_i;               //Adiciona o pedido à lista de pedidos
5.    id_req ← fedReq_i.id_req;                         //Extrai o identificador da requisição
6.    id_ui ← fedReq_i.id_Ui;                           //Extrai o identificador do usuário
7.    IdP ← DiscoverIDP(id_ui);                         //Identifica o IdP
8.    authReq_i ← <AUTHREQ, id_ui, id_req, sign_F, cert_F>;  //Gera um pedido de autenticação
9.    send authReqi to IdP;                             //Envia o pedido ao IdP do usuário
10. upon Receive authToken_i from IdP                   //Ao receber um token de autenticação
11.   token_ui ← authToken_Ui.token_ui;                 //Extrai o token
12.   id_req ← authToken_i.id_req;                      //Extrai o identificador da requisição
13.   if verifyToken(token_ui) ∧ fedReqList.get(id_req) then  //Se o token for válido e
14.     broker_Ui ← createBroker(fedReq_i, token_ui);   //Instância um broker
15.   end if;
```

c) *Broker*

O *broker* é o agente instanciado com o objetivo da negociação e alocação de recursos na federação para cada contrato de usuário. Este *broker* é ativo durante o ciclo de vida de um contrato e finalizado quando este contrato expirar. O algoritmo 3 descreve o comportamento do *broker* na busca e gerência de um *SLA* de usuário. Para a instanciação de um *broker* (linha 8 - Algoritmo 3), o *SF* recebe como parâmetro a requisição do usuário, assim como o *token* gerado pelo *IdP* no processo de autenticação. Este *token* será usado posteriormente nos controles de autorização quando da requisição de recursos em provedores das *clouds* da federação.

O próximo passo deste *broker* recém-instanciado é então encaminhar a requisição $fedReq_iB_i$, em nome de seu usuário, ao Painel de Recursos (*PR*) (linha 14). Estas mensagens incorporam o identificador do *broker* e a requisição do usuário. Vale ressaltar que a assinatura desta mensagem é feita com a chave privada do Suporte de Federação (*SF*) e seu certificado é enviado juntamente com esta requisição de modo que esta assinatura possa ser validada.

A requisição $fedReq_iB_i$, com as solicitações do usuário, é então enviada pelo *broker* ao *PR* (linha 15, Algoritmo 3). Como consequência, um temporizador é iniciado pelo *broker* para controlar em prazo estipulado as recepções de respostas a esta requisição (linha 16). No caso de decorrência deste prazo (linhas 44 – 47), a mensagem $fedReq_iB_i$ é enviada novamente ao Painel de Recursos e um novo prazo é estimado para o recebimento das respostas correspondentes.

As respostas à requisição $fedReq_iB_i$ que chegam ao *broker* vêm diretamente dos provedores de *cloud*. No recebimento destas mensagens de resposta ($fedRes_i$), o *broker* verifica se o provedor correspondente pode atender totalmente os recursos solicitados na requisição do usuário (linhas 17-19). Em caso afirmativo, o temporizador iniciado para espera de respostas é removido e uma resposta $fedReq_iAck_i$ é enviada a este provedor (linhas 21-23), informando a concordância em alocar todos os recursos indicados.

Algoritmo 3 - *Broker*

Algoritmo 3. *Cloud Broker* B_i

```
1:  Init:
2:  resReq ← Ø;                                   //Recursos recebidos
3:  partFedRes ← Ø;                               //Recursos Parcialmente Alocados
4:  authTknUi ← Ø;                                //Token de Autenticação
5:  fedAlRes ← Ø;                                 //Lista de recursos alocados
6:  timeout ← initialValue;                       //Temporizador de pedidos
7:  δB ← Ø

8:  upon CreateBroker(fedReqi.tokenUi) do         //Ao ser instânciado o broker
9:    idU ← fedReqi.idreq.idui;                   //Extrai o identificador do usuário
10:   idBi ← <IDBI, Bi, idU>;                     //Extrai o identificador de broker
11:   idreq ← fedReqi.idreq;                      //Extrai o identificador da requisição
12:   resReq ← fedReqi.resReqi.res;               //Extrai a lista dos recursos requisitados
13:   authTknUi ← tokenUi;                        //Extrai o token de autenticação
14:   fedReqiBi ← <FEDRB, idBi, idreq, fedReqi, signF, certF>;   //Gera um pedido de recursos
15:   send fedReqiBi to RP;                       //Envia o pedido para o RP
16:   δB.time ← time();                           //Inicia um temporizador

17: upon Receive fedResi from Cj do               //Ao receber resposta dos provedores
18:   idreq ← fedResi.idreq;                      //Verifica o id.
19:   if fedResi.canff = "Full" then              //Se o atende totalmente
20:     nRes ← "Full";                            //Marca a resposta como full
21:     δB.time ← ⊥;                              //Desativa o temporizador
22:     fedReqiAcki ← <FEDREQACK, idBi idreq, fedResi, Full, ↩   //Gera confirmação para o provedor
23:         signF, certF>;
24:     send fedReqiAcki to Cj;                   //Envia confirmação para o provedor
25:   else                                        //Caso contrário
25:   partFedRes ← partFedRes ∪ fedResi;          //Adiciona o atendimento parcial a lista
26:   fedResCount ← fedResCount + fedResi.MaxFreeRes;   //Adiciona os recursos recebidos
27:   if (fedResCount ≥ | resReq|) then           // Verifica se obteve todos recursos
28:     δB.time ← ⊥;                              //Desativa o temporizador
29:     orderByMaxFreeResources(partFedRes);      //Ordena pelo mais livre
30:     for all Cj ∈ partFedRes                   //Todas respostas
31:       nRes = [i...n];                         //Seleciona quais recursos deseja alocar
32:       fedReqiAcki ← <FEDREQACK, idBi, idreq, ↩   //Gera Confirmação
              fedResi, nRes, authTknUi, signF, certF>;
33:       send fedReqiAcki to Cj;                 //Envia confirmação para o provedor
34:     end for;
35:   end if;
36: end if;

37: upon Receive fedAllocAcki do                  //Ao receber confirmação de alocação
38:   idreq ← fedAllocAcki.idreq;                 //Extrai identificação de requisição
39:   fedAlRes ← fedAlRes ∪ fedAllocAckj.resPointers;   //Adiciona a lista de ponteiros de recursos
40:   if (|fedAlRes| ≥ |resReq|) then             //Se recebeu todos os ponteiros de recursos
41:     fedRepi ← <FEDREPLY, idBi,idreq, fedAllocRes, signF,   //Gera resposta para usuário
          certF>;
42:     send fedRepi to Ui;                       //Envia resposta com ponteiros para usuário
43:   end if;

44: upon (time() - δB.time) ≥ timeout do          //Se o temporizador exceder o tempo de espera
45:   send fedReqiBi to RP;                       //Envia um novo pedido ao Painel de Recursos
46:   δB.time ← time();                           //Renova o temporizador
47:   'timeout ← NewEstimation(timeout);          //Estima um novo período para resposta
```

Caso a resposta do provedor não indique o atendimento total da requisição do usuário, o *broker* então armazena esta resposta para uma possível alocação distribuída entre os diversos provedores de *cloud* da federação (linhas 24 - 26). Quando a soma dos recursos que podem ser suportados parcialmente pelos provedores de *cloud*, for suficiente para o atendimento das necessidades do usuário (linha 27), o *broker* então remove o temporizador (linha 28) e ordena a lista de provedores que atenderam a solicitação de recursos via Painel (linha 29). Esta ordenação usa o critério de número máximo recursos livres por provedor de *cloud*, tentando com isto minimizar o número de provedores que juntos atendam a requisição do usuário.

Após essa ordenação, é enviada para cada um dos provedores uma mensagem $fedReq_iAck_i$ para que os recursos sejam definitivamente alocados nos provedores. É importante ressaltar que no caso de alocações parciais, para cada uma das mensagens $fedReq_iAck_i$ enviadas, o *broker* informa a quantidade de recursos que cada um dos provedores deve alocar para o usuário solicitante. Esta informação é importante, uma vez que o número de recursos que é solicitado para alocação em cada provedor não é necessariamente o que cada provedor tinha reservado anteriormente para a requisição.

Efetuado o envio da confirmação para que os recursos sejam alocados, o *broker* então fica no aguardo de mensagens $fedAllocAck_i$. Nestas mensagens são enviados os ponteiros (endereços) que indicam onde os recursos estão disponíveis para o usuário. Com o recebimento destas mensagens $fedAllocAck_i$ (linha 37, Algoritmo 3), o *broker* armazena os ponteiros em uma lista e verifica se o número de ponteiros recebidos atende ao número de recursos da requisição do cliente (linha 40). Caso a verificação indique a suficiência de recursos, uma mensagem $fedRep_i$ é gerada com os ponteiros dos recursos (linha 41) e enviada para o *Cloud User* (linha 42, Algoritmo 3), finalizando o processo de aquisição de recursos.

d) *PR* (Painel de Recursos)

Como citado anteriormente, um serviço de eventos especializado, nomeado como Painel de Recursos (*PR*), foi proposto em nosso modelo de federação, permitindo com isto que os provedores de *cloud* sejam notificados sobre demandas de recursos. O comportamento do *PR* é explicitado no Algoritmo 4. Ao receber uma requisição $fedReq_iB_i$ (linha 8, Algoritmo 4), este serviço (*PR*) deve notificar aos provedores apropriados sobre a demanda correspondente de recursos. O *PR* percorre então as listas associadas com os provedores que podem suprir as demandas de recursos do usuário (linhas 12 - 23, Algoritmo 4), enviando

mensagens $reqNotif_i$ para todos os provedores que oferecem os tipos determinados de recursos.

Algoritmo 4 – Painel de Recursos

Algoritmo 4. *ResourcePanel RP*

```
1.  Structures
2.    Member{                                          //Define estrutura de membros, com seus
3.      Type} // VmTypeA, VmTypeB                      //respectivos recursos oferecidos
4.      Name[]}; //C1, C2, C3....
5.  Init:
6.  memList[] ← Members;                               //Lista de Membros da Federação
7.  reqList[] ← Ø;                                     //Lista de requisições

8.    upon Receive fedReq_iB_i from B_i                //Ao receber uma requisição de um Broker
9.      id_req ← fedReq_iB_i.id_req;                   //Extrai a identificação da requisição
10.     id_RP ← <IDRP, idRP, idReqP>;                  //Gera um identificação
11.     reqList ← reqList ∪ fedReq_iB_i;               //Adiciona a lista de requisições
12.     for all Type ∈ fedReq_iB_i.resReq_i.res.Type do //Para todos os tipos
13.       case Type is                                 //Se o tipo for
14.         VmTypeA:
15.             type ← VMTypeA;
16.             reqNotif_i ← <REQNOTIF, id_RP, id_req, fedReq_iF, type, sign_rp, cert_rp>;  //Gera notificação
17.             send reqNotif_i to all C_j ∈ MemList.Type(VmTypeA);                          //Envia notificação
18.         VmTypeB:
19.             type ← VMTypeB;
20.             reqNotif_i ← <REQNOTIF, id_RP, id_req, fedReq_iF, type, sign_rp, cert_rp>;  //Gera notificação
21.             send reqNotif_i to all C_j ∈ memList.Type(VmTypeB);                          //Envia notificação
22:       end case
23:     end for

24:     upon Receive memEnter_m from C_j               //Pedido de entrada de membro
25:       case memEnter_j.Type is                      //Verifica o tipo de recurso
26:         VmTypeA:
27:           memList.VmTypeA ← memList.VmTypeA ∪ memEnter_m.C_j;   //Adiciona em sua respectiva lista
28:         VmTypeB:
29:           memList.VmTypeB ← memList.VmTypeA ∪ memEnter_m.C_j;   Adiciona em sua respectiva lista
30:         ...
31:       end case

32:     upon Receive memExit_m from C_j                //Ao receber um pedido de saída
33:       case memExit_j.Type is                       //Verifica o tipo de recurso
34:         VmTypeA:
35:           memList.VmTypeA ← memList.VmTypeA - memEnter_m.C_j;   //Remove o membro da lista
36:         VmTypeB:
37:           memList.VmTypeB ← memList.VmTypeB - memEnter_m.C_j;   //Remove o membro da lista
38:         ....
39:       end case
```

Diante da disponibilidade ou indisponibilidade momentânea de recursos, os provedores de *clouds* da federação podem se inscrever ou pedir para serem removidos destas listas associadas a tipos de recursos. Ou seja, estas listas evoluem refletindo a dinamicidade da federação em termos da disponibilidade de recursos. O Painel de Recursos oferece duas operações em suas interfaces que permitem provedores entrarem ou saírem destas listas associadas a tipos: $memEnter_m$ e $memExit_m$. O *PR*, ao receber uma mensagem $memEnter_m$, adiciona o provedor em listas (associadas) de tipos de recursos dos quais este provedor dispõe, deixado o mesmo habilitado para receber notificações correspondentes (linhas 25 - 32). No recebimento de uma mensagem $memExit_m$, o *PR* remove o provedor das listas associadas de tipos de recursos que o mesmo já não dispõe (linhas 34 – 40).

e) *Cloud Provider*

Provedores de *cloud* são os fornecedores de recursos na federação. Uma vez estabelecidos os contratos (via *brokers*), os usuários passam a ter acesso direto aos recursos alocados, segundo os *SLAs* estabelecidos. O algoritmo 5 descreve os provedores de *clouds* nas suas interações na alocação de recursos.

Ao receber uma notificação ($reqNotif_i$) do Painel de Recursos, o provedor de *cloud* primeiramente recupera o contrato de *SLA* da requisição do usuário e verifica se o mesmo está de acordo com os parâmetros e requisitos definidos para o fornecimento de recursos (linhas 9-14, algoritmo 5). Caso o provedor possa atender aos requisitos do *SLA*, a disponibilidade dos recursos solicitados é verificada (linha 15).

O atendimento pelo provedor de uma solicitação de recursos pode ocorrer de duas maneiras: de forma total, onde o provedor pode atender toda a demanda do usuário, ou parcial, quando somente parte dos recursos requisitados estão disponíveis no provedor. Se a quantidade de recursos no provedor é suficiente para atender por completo a solicitação (linha 16), o provedor de *cloud* faz então a reserva destes recursos (linha 17), e monta a mensagem $fedRes_i$, informando o atendimento integral da solicitação do usuário pelo provedor (*canff* ← *Full*). Esta mensagem, que é uma resposta à notificação $reqNotif_i$, é enviada diretamente ao *broker* solicitante (linhas 19-20). Um temporizador é iniciado após o envio de mensagem $fedRes_i$ (linha 21) para controlar um prazo definido segundo políticas do provedor de *cloud*. Este prazo é usado para liberar os recursos reservados, caso não haja resposta de confirmação do *broker* dentro do limite de tempo estipulado (linhas 45 – 46).

No caso da impossibilidade de atendimento total da requisição do usuário, o provedor monta sua mensagem $fedRes_i$, informando o atendimento somente parcial da solicitação (*canff* ← *Partial*) e também a quantidade disponível (*maxFreeRes*) destes recursos (linhas 22-26). Neste último caso, os recursos são também reservados e um temporizador é iniciado para que os recursos sejam liberados quando não houver resposta a esta reserva no prazo estabelecido (linha 28). No recebimento de uma mensagem $fedReq_iAck_i$ (linha 31), o provedor de *cloud* recupera e valida o *token* de autenticação do usuário (linhas 33-34).

Esta validação envolve a verificação dos dados de assinatura do *token* (data de emissão e duração do mesmo), dentre outros atributos do usuário contidos neste *token*, e corresponde ao controle de autorização de

um provedor de *cloud*[5]. A apresentação do *token* corresponde à posse de uma competência (ou *capability*) na nomenclatura de controle de acesso (LANDWEHR, 2001).

Com a validação do *token*, o provedor remove o temporizador que controla a reserva e os recursos referentes à requisição são então alocados ao usuário (linhas 35 - 40). É importante ressaltar que, no caso de atendimento parcial, o provedor de *cloud* não necessariamente aloca todos os recursos previamente reservados (*maxFreeRes*, na linha 26). O *broker*, no envio de sua mensagem $fedReq_iAck_i$, define a quantidade de recursos que deseja no atendimento parcial de cada provedor selecionado. Após alocar os recursos, o provedor de *cloud* responde ao *broker*, através da mensagem $fedAlAck_j$, indicando os ponteiros (endereços) dos recursos alocados (linhas 41-42).

Os provedores devem também gerenciar a evolução dinâmica da disponibilidade de recursos. Com o decorrer do tempo, tipos de recursos podem ficar indisponíveis ou, ainda, disponíveis nestes provedores. Diante destas variações, os provedores podem remover suas inscrições no Painel de Recursos (*PR*) para receberem notificações sobre determinados tipos de recursos; assim como podem se inscrever em listas associadas quando os recursos correspondentes estiverem novamente disponíveis. Para estas evoluções de seus registros em listas associadas de tipos de recursos, o provedor dispõe dos envios das mensagens $memEnter_m$ e $memExit_m$ para ser adicionado ou removido de listas associadas no *PR* (linhas 47 - 58).

f) *IdPs* (Provedor de Identidades)

A autenticação de usuários é importante para o acesso ao Suporte da Federação (*SF*), bem como para a alocação de recursos junto aos provedores de *Cloud*. Diante disto, o nosso modelo adota um gerenciamento de identidades centralizado (JØSANG et al., 2005), onde provedores de identidades autenticam os usuários dos serviços para o *SF* e os provedores de *cloud*. O algoritmo 6 mostra as interações de um *IdP* da federação no modelo proposto. Mais detalhes sobre os processos de autenticação serão descritos nas seções 4 e 6, onde são definidos os provedores de identidades, suas estruturas e especificações de

[5] Os controles de autorização podem ser mais sofisticados e pedir mais atributos do usuário, seguindo as políticas definidas no provedor de *cloud*. Estes atributos seriam enviados em asserções de atributos, em comunicações entre o provedor e o *IdP* do usuário. Omitimos esta possibilidade tornando nosso modelo de autorização muito simples.

funcionamento. Neste capítulo nos atemos ao papel que o provedor de identidades desempenha na federação.

Ao fazer o seu primeiro acesso ao *SF*, o usuário é direcionado ao seu *IdP* para se autenticar. Para isto, o *SF* envia uma requisição de autenticação ($authReq_i$) ao *IdP* correspondente (linhas 8 e 9,

Algoritmo 2). Este *IdP*, ao receber esta mensagem (linha 3, Algoritmo 6), verifica se o usuário não está autenticado, através da existência do *token* de autenticação correspondente em sua lista de *tokens* ativos (linha 7, Algoritmo 6). Caso o usuário já tenha sido autenticado previamente, o *IdP* responde ao *SF* enviando o *token* recuperado (linha 8). Caso o usuário ainda não tenha se autenticado, o *IdP* requisita então que este informe suas credenciais (certificados, usuário/senha) enviando uma mensagem $authCReq_i$ ao *Cloud User* (linhas 10 e 11, Algoritmo 6). No recebimento das credenciais do usuário, através da mensagem *authCredi*, uma vez validadas estas credencias (linhas 12-14), o IdP gera um token de autenticação devidamente assinado ($authToken_i$), o armazena e, por fim, envia uma cópia do mesmo para o Suporte de Federação que pediu a autenticação do usuário (linhas 15 - 19, Algoritmo 6).

3.2.3 Considerações sobre os Algoritmos

Algumas considerações devem ser feitas sobre as operações efetuadas pelo *broker* (Algoritmo 3). Em nossa abordagem, no caso de somente um provedor atender a requisição do usuário, não entramos em méritos se este provedor é o que melhor atende a requisição (em termos de QoS, ou outro fator). Simplesmente o primeiro provedor de *cloud* que atende a requisição completamente é o escolhido (linhas 17-23, algoritmo 3). Um modelo que aguardasse as diversas respostas e analisasse as mesmas, usando critérios de custo ou qualquer outro, poderia gerar um melhor benefício para o usuário da federação. Porém, neste momento, este não foi o foco de nosso trabalho.

A segunda consideração a ser feita é sobre o atendimento da requisição do usuário por diversos provedores de *cloud* (linhas 24 - 33, algoritmo 3). Neste caso, por simplicidade, adotamos uma heurística para ordenação da lista de provedores que devem atender a requisição do usuário, priorizando os com maior número de recursos disponíveis. Podem ocorrer situações onde são excluídos provedores que podem atender totalmente a requisição. É o caso quando a soma dos recursos parciais é atingida antes do recebimento de uma resposta com a possibilidade citada. Assim como em nossa primeira consideração sobre o modelo, a aplicação de uma heurística mais complexa que analisasse os diversos parâmetros das respostas parciais poderia trazer benefícios.

Algoritmo 5 – Cloud Provider

Algoritmo 5. *Cloud C_j*

```
1.  Init:
2.  resReq[] ← Ø;                                   //Lista de recursos
3.  localRes[] ← Ø;                                 //Lista de recursos locais
4.  resPointers[] ← Ø;                              //Lista dos ponteiros
5.  fedReqAck[] ←Ø;                                 //Lista das confirmações
6.  timeout ← initialValue;                         //Definição de tempo de espera
7.  available ← True;                               //Disponibilidade
8.  δi ← Ø;                                         //Temporizador

9.  upon Receive reqNotifi from RP do               //Ao receber uma notificação do RP
10.   slau ← reqNotifi.fedReqiF.fedReqi.slai;       //Extrai o SLA
11.   type ← reqNotifi.type;                        //Extrai o tipo de recurso requisitado
12.   resReq ← reqNotifi.fedReqiBi.fedReqi.res(type); //Extrai a quantidade de recurso
13.   idreq ← reqNotifi.idReq;                      //Extrai a identificação da requisição
14.   if agreeSLA(slau) then                        //Se puder cumprir os termos do SLA
15.     localRes ← getFreeResources(resReq, type);  //Recupera a qtde. de rec. livres
16.     if |localRes| ≥ | resReq| then              //Se a qtde. for suficiente para atender
17.       reserveRes(idreq, localRes);              //Faz a reserva dos recursos
18.       canff ← Full;                             //Define como atendimento total
19.       fedResi ← <FEDRES, idCj, idreq, canff, signcj, certcj>; //Cria resposta a requisição
20.       send fedResi to Bi;                       //Responde a requisição para o broker
21.       δi.time ← time();                         //Inicia um temporizador para resposta
22.     else                                        //Caso contrário
23.       canff ← Partial;                          //Marca como atendimento parcial
24.       reserveRes(reqID, localRes);              //Faz a reserva dos recursos
25.       maxFreeRes ← |localRes|;                  //Define a qtde. de recursos livres
26.       fedResi ← <FEDRES, idCj, idreq, canff, maxFreeRes,
      signcj, certcj>;                              //Cria resposta a requisição
27.       send fedResi to Bi;                       //Responde a requisição para o broker
28.       δi.time ← time();                         //Inicia um temporizador para resposta
29.     end if
30.   end if

31. upon Receive fedReqiAcki from Bi                //Ao receber uma confirmação do Broker
32.   idreq ← fedReqiAcki.idreq;                    //Extrai o identificador da requisição
33.   tokenui ← fedReqiAck.authTknUi;               //Extrai o token de autenticação
34.   if validadeToken(tokenui) then                //Se o token de autenticação for válido
35.     δi.time ← ⊥;                                //Remove o temporizador
36.     idBi ← fedReqiAcki.idBi;                    //Extrai a identificação do broker
37.     if (resReq = "Full") then                   //Se o atendimento for total
38.       resPointers ← Allocate(idreq);            //Aloca os rec. e retorna os ponteiros
39.     else                                        //Caso Contrário
40.       resPointers ← Allocate(idreq [i…n]);      //Aloca os recursos parciais
41.       fedAlAcki ←<FEDALCK, idCj, idBi, idreq, resPointers,
      signcj, certcj>;                              //Gera mensagem de resposta
42.       send fedAlAckj to Bi;                     //Envia para o broker
43.     end if;
44.   end if;

45.   upon (time() - δi.time) ≥ timeout do          //Se a temporização se esgotar
46.     freeResource(idreq);                        //Libera os recursos reservados

47. upon LocaResource.Type.count ≤ 0 do             //Ao se esgotarem os recursos
48. if available = True then                        //Se for um provedor disponível
49.   memExitm ← <MEMEXIT, Cj, Type>                //Cria msgm. para sair dos membros
50.   available ← False;                            //Se marca como indisponível
51.   send memExitm to RP                           //Envia mensagem para o RP
52. end if

53. upon LocaResource.Type.count > 0 do             //Ao disponibilizar recursos
54. if available = False then                       //Se não estiver disponível
55.   memEnterm ← <MEMENTER, Cj, Type>              //Cria msgm. para entrar dos membros
56.   available ← True;                             //Se marca como disponível
57.   send memEnterm to RP;                         //Envia mensagem para o RP
58. end if;
```

Algoritmo 6 – Provedor de Identidades

Algoritmo 6. *Identity Provider IdP*

```
1.  Init:
2.  tokenList[];                                   //Lista dos tokens disponíveis

3.  upon Receive authReq_i from F                  //Ao receber uma req. de autenticação
4.    U_i ← authReq_idp.id_Ui;                     //Extrai o usuário
5.    id_req ← authReq_idp.id_req;                 //Extrai o identificação da requisição
6.    authToken_i ← tokenList.get(U_i);            //Busca o token para o usuário em lista
7.    If authToken_i ≠ null then                   //Caso encontre um token
8.      send authToken_i to F;                     //Envia token de autenticação para F
9.    else                                         //Caso Contrário
10.     authCReq_i ← <AUTHCREQ, id_IdP, id_req, Sign_IdP, Cert_IdP>;   //Cria pedido de autenticação
11.     send authCReq_i to U_i;                    //Envia pedido para o Usuário

12: upon Receive authCred_i from U_i               //Ao receber credenc. do usuário
13:   credentials ← authCred_ui.credentials;       //Extrai as credenciais
14:   if verifyCredentials(credentials) then       //Verifica a validade
15:     U_i ← authCred_i.id_Ui;                    //Extrai o idenficador de usuário
16:     id_req ← authCred_i.id_req;                //Extrai o identificador da req.
17:     authToken_i ← <AUTHTOKEN, id_Ui, id_req, signature,   //Gera um token de autenticação
        RevokeDate, sign_idp, cert_idp>;
18:     tokenList ← tokenList ∪ authToken_i;       //Adiciona o token a lista
19:     send authToken_i to F;                     //Envia o token para F
20:   end if;
```

A mensagem *fedReq*$_i$*Ack*$_i$ enviada pelo *broker*, definindo a alocação desejada nos provedores de *cloud* escolhidos (linha 13, Algoritmo 3), possui um campo importante que comporta o *token* de autenticação do usuário (*authTkn*$_{Ui}$). Este *token* serve como uma *capability*, permitindo que o usuário reconhecido e aceito na federação tome posse dos recursos solicitados (controle de autorização no provedor de *cloud*). Neste modelo, não trabalhamos com permissões de granularidade mais finas, mas políticas de autorização mais sofisticadas podem ser definidas com o uso de atributos de usuários disponíveis no *IdP* (*ABAC*: *Attribute Based Access Control* (JIN; KRISHNAN; SANDHU, 2012)).

Na verificação de *SLAs* de usuário, não entramos em discussões sobre negociações dos parâmetros descritos neste contrato. Em nossa proposta o provedor de *cloud* aceita ou não o *SLA* do usuário (linha 14, Algoritmo 5). Porém um protocolo que levasse em conta uma negociação entre o que pode ser oferecido pelo provedor e as necessidades de usuário poderia tornar esta relação de uso de recursos mais flexível.

Outro ponto a ser considerado é a reserva de recursos no provedor de *clouds* (linhas 17 - 28, algoritmo 5). No momento da reserva, o provedor deixa de oferecer estes recursos a outras demandas, mesmo que ainda não alocados. Estes recursos ficam indisponíveis para futuras demandas por certo prazo controlado a partir da estrutura δ_i*.time* (linha 28, algoritmo 5) e controlado pela rotina descrita nas linhas 45 e 46 do

algoritmo 5. Uma vez ultrapassado este prazo sem alocação, os recursos voltam a ficar disponíveis para futuras demandas. Nesse sentido, a definição deste prazo deve levar em conta um *tradeoff* entre poder contribuir com o atendimento aos contratos e não perder outras demandas.

A nossa proposta adota um modelo de gerenciamento de identidades na federação, ao contrário de muitos trabalhos da literatura, onde as identidades de usuários são tarefas locais de cada provedor de *cloud*. Embora em nossa proposta não seja afastado o uso de controles locais (e das correspondentes bases de dados de atributos) na autenticação de usuários em alguns provedores de *clouds*, estas autenticações locais não permitem que recursos sejam buscados na federação. Usuários, para o acesso a recursos na federação, devem ter registros e executar o *login* em *IdPs* da federação. Provedores de *cloud* que delegarem totalmente a autenticação para um *IdP* externo aos seus domínios locais, não precisam manter bases de dados complexas, com os atributos de usuários.

A autenticidade do usuário na nossa proposta é baseada nas verificações da plataforma *OpenID* de gerenciamento de identidades (OPENID, 2014). O *OpenID* foi adotado em nossos trabalhos devido à simplicidade de seus protocolos e porque nos últimos anos tem se transformado em um padrão de fato, com várias grandes empresas aderindo a esta especificação. Embora citado muitas vezes na literatura como implementando o modelo de identidades *federadas* (JØSANG et al., 2005) o *OpenID* implementa o modelo c*entralizado* de gerenciamento de identidades, uma vez que *IdPs Open ID* não mantêm relações de confiança entre si. Mesmo assim, o *OpenID* apresenta facilidades com mecanismos simples para a procura do *IdP* de um usuário e a possibilidade *Single-Sign-On*, onde um usuário já autenticado pode acessar vários serviços com a mesma autenticação por um certo período.

3.3 SIMULAÇÕES E ANÁLISES

O objetivo central dos testes realizados de simulação é obter medidas aproximadas dos custos em mensagens e do tempo necessário para alocar recursos (máquinas virtuais) na federação. Para tanto, um modelo de simulação foi implementado, conforme ilustrado na Figura 9. Este modelo teve como base o simulador *CloudSim* (CALHEIROS et al., 2011). Cada entidade do modelo de federação foi mapeada em uma *CloudSim Entity*, representando os diferentes atores da federação, e as trocas de mensagens foram mapeadas em *CloudSim Events*.

Cada provedor de *cloud*, representado por uma *CloudSim Entity* no modelo, é considerado como composto por diversos *hosts* que, por sua vez, são compostos por diversos *Processing Elements* (*PEs*). Para a aplicação do modelo proposto, recursos como Máquinas Virtuais (*VMs*)

foram generalizados, de forma que cada *VM* ocupe um *Processing Element* de cada *Host*.

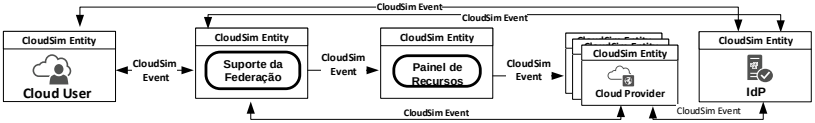


Figura 9 – Visão Geral do Modelo de Simulação.

O modelo descrito pelos algoritmos apresenta trocas de autenticação. Devido à dificuldade de representar a interação entre usuário e *IdP* no processo de *login*, no modelo simulado consideramos que o usuário já fez o seu *login* na federação. Os pedidos de autenticação enviados ao *IdP* pelo Suporte de Federação ($authReq_i$ nas linhas 8 e 9 do

Algoritmo 2) fazem parte das simulações. A resposta a estes pedidos é sempre o envio pelo *IdP* do *token* de autenticação ($authToken_i$, na linha 10 do algoritmo 2), considerando sempre que o usuário já foi previamente autenticado.

3.3.1 Definição dos parâmetros do modelo

Para executar a simulação, vários parâmetros tiveram que ser definidos. Para cada um dos eventos trocados entre as entidades foi atribuído um valor de *delay*, representando os tempos nas trocas de mensagens. A obtenção destes valores de *delay* tomou como base a nossa experiência com a *Azure Cloud* da Microsoft. Várias medições de largura de banda e latência entre servidores de diferentes regiões geográficas foram realizadas. Utilizamos instâncias de servidores das regiões: América do Sul (BRZ), Europa (EUR), Estados Unidos (USA) e Sudeste da Ásia (ASI).

Outro parâmetro usado em nossas simulações é a quantidade de recursos (*VMs*) requeridos pelos usuários que denominamos de *workload*. Para tanto, duas abordagens foram definidas: a primeira com o *workload* crescendo de forma linear e a outra com o *workload* na forma de uma distribuição aleatória. O *workload* com crescimento linear é constituído de provedores nos quais há um número fixo e conhecido de recursos livres. Este *workload*, por testes, apresenta um aumento gradativo, caracterizando um crescimento linear das demandas e definindo também o aumento de provedores atuando sobre as requisições de usuário. No outro modelo de *workload*, os recursos livres, embora fixos no sistema, são apresentados distribuídos de forma aleatória entre os diversos provedores de *cloud* da federação. O atendimento das requisições de usuários vai depender da instância de teste. Uma demanda pode ser

resolvida com poucos provedores de *cloud*, em outra instância essa mesma demanda pode envolver um número muito maior de provedores.

Um parâmetro também importante nas simulações realizadas foi o tempo aproximado para alocar recursos em um provedor de *cloud*. Usamos novamente o *Azure Cloud* na determinação do tempo para alocar *VMs* do tipo mais básico, e esta medição resultou a um valor de aproximadamente dois (2) minutos. Devido à diferença na ordem de grandeza deste tempo se comparado aos tempos das trocas de mensagens (milissegundos), não incluímos este valor nos gráficos dos resultados. Desta forma, os gráficos dos resultados apresentam os tempos resultantes das trocas do protocolo sem o tempo de instanciação dos recursos.

Por fim, definimos diferentes perfis de simulações que descrevem o posicionamento (geográfico) de cada uma das entidades da simulação. A Tabela 6 abaixo apresenta alguns dos perfis de simulação introduzidos para realização dos testes. Embora diferentes perfis tenham sido simulados, nos limitamos à apresentação de somente três perfis mais significativos em termo de diversidade de distribuição geográfica dos componentes.

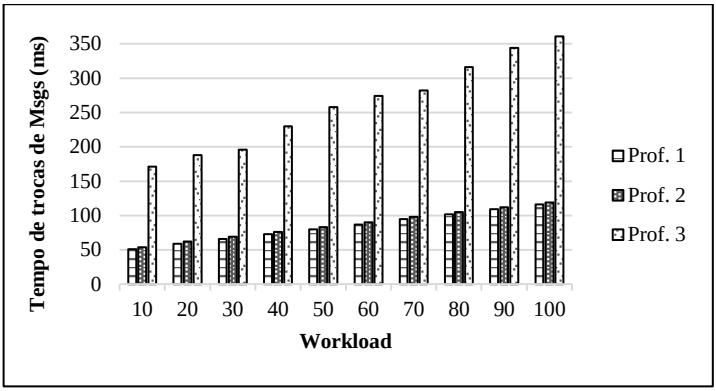
Tabela 6 – Perfis de Simulações

Perf.	**User**	**Sup.**	**Painel**	***IdP***	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**
1	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ
2	BRZ	USA	EUR	BRZ	BRZ	USA	EUR	ASI	USA	EUR	BRZ	ASI	BRZ	BRZ
3	USA	EUR	ASI	USA	BRZ	USA	EUR	ASI	BRZ	USA	EUR	ASI	BRZ	USA

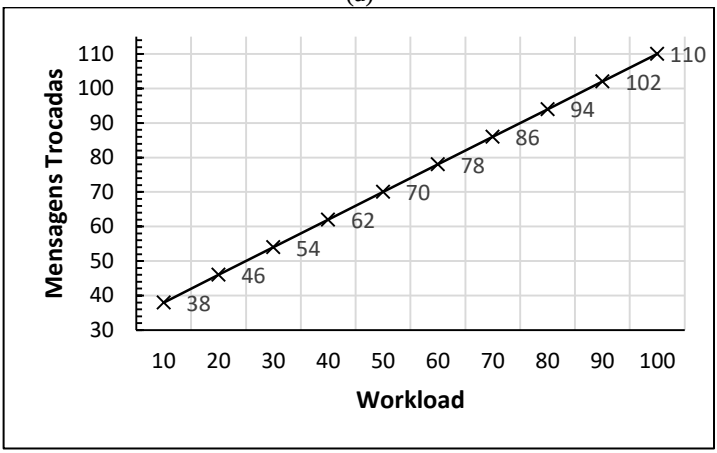
3.3.2 Modelo de Workload Linear

Neste cenário, o parâmetro *workload* foi definido em forma crescente com a inclusão de um provedor de *cloud* a cada instância de teste, compondo então uma faixa de recursos que vai de 10 até 100 *VMs*, assumindo que cada provedor sempre fornece o valor fixo de 10 *VMs* disponíveis.

A Figura 10(a) apresenta os resultados de tempo de resposta para o usuário obter recursos na federação (com o mesmo recebendo os ponteiros de recursos) em cada um dos *workloads* lineares executados nos diferentes perfis. Logicamente a localização das entidades do modelo influencia de maneira significativa no tempo de alocação de recursos na federação. No Perf.1, no qual todas as entidades do modelo fazem parte da mesma região geográfica, o tempo de resposta é menor se comparado com outros perfis, onde a distribuição das entidades envolve diferentes regiões.



(a)



(b)

Figura 10 – Resultados do *Workload* Linear

Já a Figura 10(b) apresenta o crescimento do número de mensagens quando do aumento na alocação de recursos através da inclusão de novos provedores visando o atendimento dos *workloads*. É possível perceber que o número de mensagens cresce de forma linear. Quando um novo provedor é adicionado para atender a um *workload*, oito (8) novas mensagens são adicionadas aos custos de alocação.

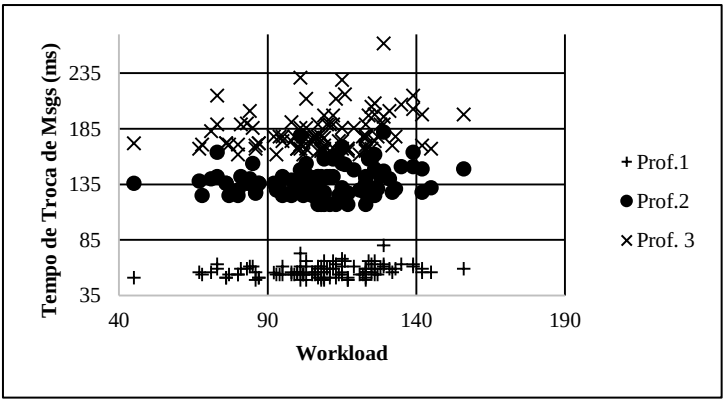
Para termos uma visão dos resultados que obtivemos no *workload* linear, tomamos como exemplo um usuário solicitando 100 recursos na federação e considerando o perfil três, que dá uma maior diversidade de distribuição geográfica do modelo. Nesta situação, considerando os resultados de nossas simulações, o tempo para alocação de recursos na federação ficou em aproximadamente 2 minutos acrescidos de 351 *ms*

consumidos em trocas de comunicações (que são seguras, envolvendo mensagens assinadas). Ou seja, se tomarmos como referência os 2 minutos que são consumidos no *Azure Cloud* para alocar *VMs*, podemos dizer que os custos de alocação em uma federação atendendo um *workload* linear são praticamente os mesmos do *Azure* e que os custos de comunicação não têm um peso significativo nestas alocações. Nas simulações foi considerado que os provedores da federação estão disponíveis para responder imediatamente às demandas de usuários.

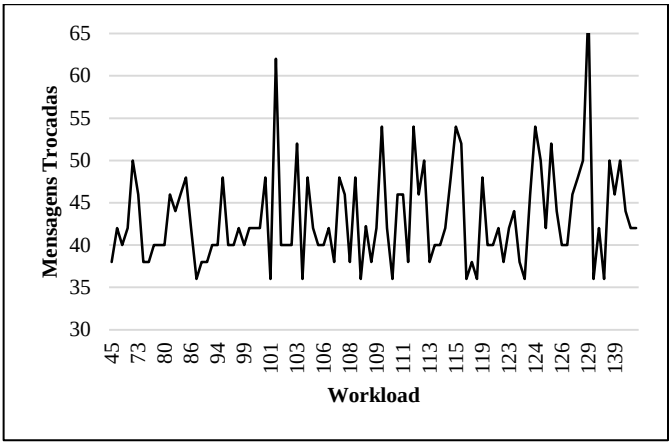
3.3.3 Modelo com *Workload* Aleatório

Nestes testes, o *workload* do usuário foi baseado em uma distribuição normal com média em 100 e desvio padrão de 20 nos recursos solicitados. A escolha desses valores não se baseou em nenhuma observação específica, mas sim na ideia de que o número de recursos requisitados pelos usuários estaria entre 80 e 120 máquinas virtuais, o que pode ser considerado como valor razoável para grandes usuários de *cloud*.

Os recursos livres por provedor são definidos com os seguintes limites: o número de *Hosts* varia entre 1 e 10 (por provedor); o número de *PEs* livres em cada *Host* tem valores entre 5 e 15; e como consequência, as *VMs* possíveis por provedor estão entre 5 e 150. A retenção destes valores para os parâmetros citados foi baseada nos inúmeros testes realizados *a priori*. Observamos que escolhas de faixas menores, por exemplo, para *Hosts* ou *PEs* resultavam em *workloads* sem solução (insuficiência de recursos), o que para nossa observação não é interessante.



(a)



(b)

Figura 11 - Resultados do *Workload* Aleatório

A Figura 11 (a) e a Figura 11 (b) apresentam os resultados de tempo de trocas e do número de mensagens trocadas em testes, considerando diferentes *workloads* aleatórios. A troca de mensagens em cada simulação pode apresentar resultados distintos, que ainda assim são considerados como válidos. Ou seja, em uma situação ideal, a execução pode envolver o menor número possível de provedores, enquanto no pior caso, a execução requer um número de provedores próximo do número de recursos requisitados pelo usuário.

Tomando a mesma situação da seção anterior, um usuário que solicita 100 recursos na federação com Perfil 3, considerando o *workload* aleatório, o tempo de alocação de recursos também ficou com um tempo próximo de 2 minutos, mas acrescido de 167 *ms* devido às trocas de mensagens. É possível perceber que a simulação com *workload* aleatório teve tempo de resposta levemente menor do que no anterior (o linear). Essa diferença se dá pelo fato de que, na simulação com *workload* linear, para a requisição de 100 recursos, obrigatoriamente o suporte irá se utilizar de todos os provedores de *cloud* da federação. Já com *workload* aleatório, a aquisição de 100 recursos se dá (na média) com o uso de um número menor de provedores de *cloud*.

3.4 LITERATURA RELACIONADA

Um dos primeiros trabalhos para federações de *clouds* foi apresentado por *Buyya* (BUYYA; RANJAN; CALHEIROS, 2010), onde uma federação de *clouds* é tida como um modelo de múltiplos provedores interagindo na busca de recursos. Esta busca é em nível provedor-provedor, não havendo um suporte global para tal. Os usuários se

associam a um provedor e este fica responsável por atender toda a demanda do cliente. Abordagens similares a esta são também encontradas em (ROCHWERGER et al., 2009, 2011), (CELESTI et al., 2010a, 2010b).

Outras experiências de federação, como as apresentadas em (CARLINI et al., 2012; COPPOLA et al., 2012; VILLEGAS et al., 2012), são próximas da nossa proposta e fazem o uso de um suporte global na busca de recursos. Estes modelos, com uma entidade central controlando buscas e alocações, faz com que usuários de *clouds* tenham também ligação com este suporte global que caracteriza a federação. Porém, apesar de não serem claramente discutidos, os registros de usuários podem estar ligados a domínios locais de seus provedores de *clouds* ou controlados a partir deste suporte global da federação. Na verdade, não existe clareza sobre como se dá o gerenciamento de identidades na maioria das abordagens citadas. Os trabalhos publicados não esclarecem se cada provedor de *cloud*, ou mesmo um suporte global, é responsável pelo seu domínio de usuários. Outra incógnita é como são executados os controles de autenticação e autorização.

Estas propostas, em sua maioria, são fundamentadas em modelo tradicional de gerenciamento de identidades, gerando certas implicações quando federações de *clouds* são definidas a partir destes controles locais de identidades. Com domínios de nomes locais e o provimento de recursos se dando a nível provedor-provedor, fica difícil tarifar o usuário pelo seu uso de recursos quando a relação entre provedores pode envolver múltiplos usuários cujos identificadores só tem sentido localmente.

Uma exceção é a abordagem descrita em (CELESTI et al., 2010b), onde os autores propõem uma federação de *clouds* baseada em relações provedor a provedor. Esta proposta considera o gerenciamento de identidades federadas. Portanto, as autenticações dos usuários em suas *clouds* locais podem ser transpostas de forma a utilizar recursos de outras *clouds* na federação, sem a necessidade da reapresentação de credenciais. Porém, os custos de manutenção de base de dados com atributos de usuários são locais.

Em (SETTE; FERRAZ, 2014), é discutido o uso de gerenciamento de identidades em *clouds*. O texto se fixa no uso de identidades federadas e a proposta apresentada não é propriamente sobre federação de *clouds*. No modelo descrito, os autores apresentam um servidor de *cloud* sendo acessado através de uma rede de confiança de *IdPs*. Esta rede de provedores de identidades (*IdPs*) é formada pelos mecanismos de autenticação do provedor de *cloud* (IdP interno ao provedor de *cloud*) e *IdPs* externos nos quais o provedor de *cloud* confia.

Este trabalho então tem seu objetivo focado em modificações estruturais em uma plataforma de *cloud* (*OpenStack*), de forma que usuários de diferentes provedores de identidades tenham suas autenticações verificadas em seus *IdPs* e aceitas na plataforma de *cloud* citada. Portanto, é um trabalho com escopo diferente do nosso. Em nosso modelo, o *IdP* é base para a identificação e os controles de segurança na federação. Ou seja, os provedores de *clouds* liberam recursos a usuários desde que estes tenham sido autenticados previamente na federação e possuam os atributos necessários para a autorização.

Outra distinção entre as abordagens da literatura é quanto à busca de recursos na federação de *clouds*. Duas abordagens podem ser distinguidas: a primeira fazendo uso de um repositório central e a outra baseada em buscas *P2P*. Nas buscas através de um repositório central (BUYYA; RANJAN; CALHEIROS, 2010), os provedores de *cloud* devem periodicamente informar quais são seus recursos disponíveis para a federação a partir deste mecanismo centralizador. Sempre que precisam de recursos, usuários devem recorrer a este repositório central. Apesar de facilitar a busca de recursos, esta técnica pode apresentar fragilidades: o repositório nem sempre apresenta uma visão consistente dos recursos disponíveis nos provedores deste sistema distribuído de larga escala que caracteriza uma federação. Isto é consequência das características destes grandes sistemas, normalmente dispersos em complexas distribuições geográficas e sujeitos a grandes demandas.

Nos modelos que fazem uso e trocas de mensagens *P2P* na busca de recursos (CELESTI et al., 2010b; COPPOLA et al., 2012) a ideia é que, sempre quando necessário, provedores de *cloud* trocam mensagens no sentido de verificar os recursos disponíveis na federação. Esta abordagem resolve o problema de consistência das informações na abordagem de repositório central, porém os custos de buscas P2P são geralmente altos em número de mensagens. E estas trocas de mensagens não garantem que os recursos estarão ainda disponíveis quando os mesmos forem definitivamente solicitados pela federação para alocação.

Em nossa abordagem propomos uma estrutura diferente, baseada em um Painel de Recursos (um serviço de eventos). Através do *PR*, um *broker* apresenta uma requisição em nome de seu usuário, e provedores de *cloud* são então notificados conforme seus registros como fornecedores dos tipos de recursos solicitados. Cada provedor decide se pode ou não atender tal demanda, respondendo diretamente ao *broker* e reservando os recursos por um tempo determinado. O uso desta técnica permite que o número de mensagens trocadas seja reduzido e garante que

os recursos sejam alocados em determinado prazo, devido a reservas feitas.

3.5 CONCLUSÃO

Neste capítulo apresentamos uma abordagem para a federação de provedores de *clouds* onde buscamos descrever modelos e algoritmos de interação entre as entidades envolvidas. Adotamos um modelo de federação que faz uso de uma infraestrutura de serviços global que é responsável pela busca e alocação de recursos em nome dos usuários na federação.

Propusemos novas abordagens neste modelo de federação. Primeiramente, quanto à busca de recursos na federação, as demandas são postas em forma de leilão em um Painel de Recursos. Os provedores somente respondem às demandas se podem atendê-las. Em caso contrário, não se manifestam nas negociações. Esta proposição torna mais simples a busca de recursos se compararmos com situações como a de repositórios globais (ou serviços de informações) que dependem de atualizações frequentes feitas pelos provedores de *cloud*. A nossa proposta garante ainda que as demandas aceitas pelos provedores sejam realmente atendidas dentro de prazos controlados.

Esta nossa proposta de federação de *clouds* serviu de veículo para os nossos estudos sobre os controles de autenticação e autorização nestes grandes sistemas. A inclusão destes controles de segurança pode ser visto como outro diferencial de nossa proposta. Dos trabalhos apresentados na literatura, somente em (CELESTI et al., 2010b) são apresentados conceitos de autenticação e autorização para acesso a recursos na federação, porém sem a definição dos protocolos necessários e suas implicações. Em nosso modelo buscamos mostrar os controles de segurança com base no modelo de gerenciamento de identidades centralizado, que foi implantado por ser mais simples e adequado à proposta de federação que descrevemos neste capítulo.

4 AUTENTICAÇÃO TOLERANTE A INTRUSÕES COM *HARDWARE* ESPECIALIZADO

O objetivo deste capítulo é apresentar um modelo de autenticação desenvolvido para ambientes de *Clouds*. Este modelo foi desenvolvido como parte da nossa contribuição no projeto *SeCFuNet*[6], que investigava aspectos de segurança de aplicações em *clouds*. Este modelo foi explorado dentro do contexto de federação de *clouds*. Neste capítulo, descrevemos as caraterísticas deste modelo de autenticação descrevendo os mecanismos que fazem do mesmo tolerante a ataques e intrusões via rede. Os resultados obtidos através de simulações do modelo são também apresentados e a análise do comportamento mesmo diante de situações de teste. Por fim apresentamos a literatura relacionada e as conclusões sobre nossos trabalhos sobre nossas proposições neste capítulo.

4.1 MOTIVAÇÃO

Em sistemas distribuídos, a aplicação de políticas e a segurança de serviços do sistema dependem, principalmente, dos controles de autenticação e autorização. A abordagem convencional empregada na concretização destes controles é baseada na implementação dos mesmos sobre provedores de serviços (*SPs*). O usuário deve se autenticar junto a um serviço e fica sujeito à política de autorização do mesmo. A complexidade das aplicações e sistemas distribuídos atuais torna este modelo limitado. Vários modelos e infraestruturas nos últimos anos têm sido introduzidos onde estes controles tomam forma a partir de um conceito mais amplo que é o de gerenciamento de identidades.

Em sistemas distribuídos de larga escala como *clouds*, grades e redes colaborativas, o gerenciamento de identidades deve se basear em uma infraestrutura que integre políticas e tecnologias, permitindo às diferentes organizações que participam destes sistemas e aplicações ultrapassar seus domínios locais (domínios administrativos ou de políticas). Diante disto, usuários e aplicações podem ter acesso, de maneira segura, a recursos remotos, desde que possuam as credenciais exigidas pelas políticas destes recursos. A infraestrutura necessária para o gerenciamento de identidades envolve muitas vezes um sistema de autenticação (usualmente em sistemas de larga escala corresponde a uma rede de autoridades de autenticação) e um sistema de gerenciamento de atributos (serviço que fornece informações adicionais sobre clientes/usuários). Estas credenciais e atributos, individual ou coletivamente, podem identificar o usuário (autenticação) e fornecer as

[6] *SecFuNet Project*: "*Security for Future Networks*" Edital Brasil/Europa, MCT/CNPq número 66/2010, processo CNPq nº 590047/2011-6.

informações necessárias (atributos do usuário) para a realização dos controles de autorização (controle baseado em atributos) permitindo assim, a execução das operações solicitadas. Em muitas propostas e infraestruturas, as autoridades de autenticação também concentram o gerenciamento de atributos e são identificados como provedores de identidades (*Identity Providers - IdP*).

Com isto então, em vários modelos de gerenciamento de identidades, a autenticação não é mais realizada nos *SPs*, mas sim em autoridades de autenticação (ou *IdPs*). A autorização continua sendo concretizada nos *SPs* porque estes conhecem seus recursos e políticas (de aplicação) e devem zelar pelos mesmos. Nesta situação, usuários e SPs não mais precisam manter sob sigilo listas muitas vezes enormes de credenciais (*senhas*, certificados, etc.). Entidades especiais são os guardiões destas informações. Nestas abordagens, a partir da identificação diante de uma destas entidades confiáveis (*IdPs* ou autoridades de autenticação), outras facilidades ficam disponíveis aos usuários do sistema como a identificação única (*Single Sign-On – SSO*) para vários acessos subsequentes (e mesmo independentes) a provedores de serviços.

Estes provedores de identidades (autoridades de autenticação) passam a ser os pontos centrais na aplicação de políticas de segurança em sistemas distribuídos. Mas, também por serem serviços que ficam disponíveis via Internet, estes *IdPs* estão sujeitos a ataques que podem resultar em intrusões nestas entidades o que seria catastrófico para a segurança das informações e recursos do sistema.

O trabalho que apresentamos neste capítulo foi parte do projeto *SecFuNet* que se propôs a desenvolver um *framework* de segurança para ambientes de *Clouds*. Este *framework* introduz, entre vários serviços, funções de autenticação e autorização para estes ambientes. O objetivo deste capítulo é descrever nossos esforços para o desenvolvimento de uma infraestrutura de gerenciamento de identidades que fosse tolerante a intrusões. Neste sentido, apresentamos o modelo desenvolvido para tornar provedores de identidades tolerantes a intrusão. O gerenciamento de identidades implementado no *SecFuNet* é fortemente dependente de componentes de *hardware* protegido. Os *logins* de usuários são feitos diretamente entre *Smartcards* (de posse do usuário) e processadores seguros dispostos em um servidor de autenticação. O gerenciamento de identidades deste projeto possui as relações de confiança entre *browsers* (usuários), *IdPs* e *SPs* fazendo uso dos protocolos do *framework OpenID* (OPENID, 2007).

Embora usualmente sejam empregados componentes seguros e protocolos que não deixam as informações de usuários disponíveis em um *IdP* invadido, estes cuidados não impedem que *IdPs* invadidos possam ter sido alvo de "implantações" de comportamentos maliciosos. Atuando segundo estes comportamentos, um *IdP* malicioso pode certificar indivíduos (intrusos) não autorizados por políticas, assim como também pode não autenticar usuários reconhecidos pelas mesmas políticas.

As soluções para tolerância a intrusões ou a faltas bizantinas BFT ("*Byzantine Fault Tolerance"*) são normalmente baseadas em replicação Máquina de Estados (ME) (SCHNEIDER, 1990). Vários trabalhos presentes na literatura apresentam soluções práticas para tolerância a faltas maliciosas (intrusões), sendo os mais conhecidos o *PBFT* (CASTRO; LISKOV, 2002) e o *Zyzzyva* (KOTLA et al., 2007). Estes trabalhos assumem que em suas replicações, enquanto os limites de faltas maliciosas (definidos como *f*) não forem atingidos, a segurança do sistema é garantida. Um dos problemas destas abordagens está no fato das mesmas usarem grande número de réplicas físicas para execução do protocolo. No *PBFT*, o número de réplicas necessárias para execução do protocolo é definido como $3f + 1$. Estes protocolos, se usados para a tolerância a intrusões de *IdPs*, devido aos altos custos em termos de mensagens, tornariam mais custoso o desempenho do procedimento de *login*.

O modelo de tolerância a intrusões que usamos neste capítulo é baseado na tecnologia de virtualização e não envolve replicação física, o que é benéfico para o nosso sistema onde os *IdPs* fazem uso de *hardware* especializado (processadores seguros) e cuja replicação aumentaria os custos financeiros e poderia também trazer novos problemas como o aumento da disponibilidade de informações sigilosas na rede. Nossa abordagem de replicação faz uso de memória compartilhada e é uma adaptação do modelo apresentado em (LAU; BARRETO; FRAGA, 2012a, 2012b). As adaptações neste modelo foram produzidas para adequá-lo às características do *IdP* e aos protocolos do *framework OpenID*.

4.2 CARACTERIZAÇÃO DO AMBIENTE DE AUTENTICAÇÃO

O processo de autenticação de usuários em sistemas distribuídos envolve, em um primeiro passo, um procedimento de *login*, onde um usuário, através de trocas com um provedor de identidades (usando técnicas como *userid*/*password*, *userid*/certificado, protocolos de *Challenge*/*Response*, etc.), tem a sua identidade validada diante do sistema. Este procedimento normalmente termina com o usuário recebendo do seu *IdP* um *token* ou asserção de autenticação que será

usados para atestar sua autenticidade diante de outras entidades do sistema em subsequentes interações.

Em sistemas de computação em nuvens, serviços de gerenciamento de identidades são usualmente implementados em *clouds* privadas ou em servidores com *hardware* especial. A não colocação de um *IdP* em *clouds* públicas se justifica pela criticidade das informações. O projeto *SecFuNet*, por enfatizar o uso de processadores seguros no processo de autenticação de usuários, apresenta limitações de escala. Como consequência, o gerenciamento de identidades neste projeto envolve a necessidade de vários *IdPs*, cada um com os seus usuários e definindo um domínio próprio de política. Estes provedores de identidades fornecem asserções de autenticação, seguindo suas políticas, para usuários e provedores de serviço (*SP*s) de seus domínios de política. Nestes domínios, portanto, o gerenciamento de identidades segue uma abordagem centralizada, onde o usuário de posse de um cartão inteligente (*smartcard*), fornece informações que são verificadas em um processador seguro no *IdP* do seu domínio. O *IdP* desempenha o papel de uma terceira parte confiável nas interações entre usuário e *SP*s. Esta abordagem centralizada é largamente usada em sistemas corporativos.

No entanto, para sistemas complexos envolvendo ambientes de *clouds*, que normalmente se estendem além de domínios locais, este modelo de intermediação simples é limitado. É necessário expandir esse modelo para um sistema de gerenciamento de identidade com base em redes de confiança, envolvendo vários destes domínios locais de identidade. Duas possibilidades são correntes neste caso: Ou os *IdPs* dos diversos domínios estabelecem entre si relações de confiança assumindo então a abordagem de gerenciamento de identidades conhecida como "identidades federadas", os *SPs* possuem listas de *IdPs* em quem confiam (rede de confiança centrada em *SPs*) e os usuários devem estar registrados em um destes *IdPs*. Esta última possibilidade de rede de confiança é construída para autenticações baseadas no modelo centralizado de gerenciamento de identidades.

A aplicação das abordagens citadas que estendem a escala das relações de confiança, é facilitada pelo de Infraestruturas de Autenticação e Autorização (*IAAs*). Exemplos notórios destas infraestruturas são: o *Shibboleth* (LEWIS, 2008) e o *Liberty Alliance* (LIBERTY, 2003) para abordagens de identidades federadas e *OpenID* como abordagem centrada em *SP*s. No projeto que estamos descrevendo, seguimos a abordagem do *OpenID*, onde cada provedor de serviços (*SP*) mantém uma lista de *IdPs* em que confia (ou seja, a rede de confiança está centrada em *SPs*). Um

usuário neste sistema deve ter a sua identidade reconhecida em pelo menos um destes *IdPs*.

4.3 VIRTUALIZAÇÃO EM ALGORITMOS TOLERANTE A INTRUSÕES

O uso da tecnologia de virtualização tem tornado possível a implementação de novos protocolos e suportes eficientes que permitem a tolerância a intrusão ((JANSEN et al., 2008; LAU; BARRETO; FRAGA, 2012b; REISER, 2007)). Este é o caso do algoritmo *IT-VM* (*Intrusion Tolerance by Virtual Machines* (LAU; BARRETO; FRAGA, 2012b)) que define réplicas de servidores implementadas isoladas em máquinas virtuais (*VMs*) e que se comunicam via memória compartilhada na efetivação de uma replicação Máquina de Estados (*ME*). As réplicas e suas *VMs* são mapeadas em somente um servidor físico. Além de evitar em muito os custos de mensagens enviadas sobre suporte de comunicação em situações da necessidade de acordo entre réplicas, este modelo usufrui de algumas vantagens da tecnologia de virtualização. Como no protocolo ZZ (WOOD et al., 2011), o *IT-VM* apresenta funções para a alteração dinâmica da replicação do modelo, ou seja, réplicas maliciosas quando detectadas, são removidas e substituídas de forma eficiente e sem deixar que a replicação durante seu ciclo de vida fique limitada a um número fixo de falhas no sistema. Mas, no protocolo *ZZ*, diferentemente do *IT-VM*, as *VMs* de réplicas são executadas em um conjunto de máquinas físicas. Com isto, no ZZ, as *VMs* não interagem via memória compartilhada mesmo quando estão dispostas na mesma máquina física. Estas interações entre *VMs* são realizadas através de trocas de mensagens via protocolos de rede, implicando custos adicionais no desempenho deste último protocolo. O *ZZ* também não trabalha com a ideia de elementos confiáveis como o modelo *IT-VM* e, portanto, não permite a separação entre faltas *crash* e de faltas maliciosas (intrusões) (VERÍSSIMO et al., 2003).

As soluções do *IT-VM* e do ZZ operam com uma replicação mínima de $f+1$ réplicas ativas na configuração quando não ocorrem falhas de réplicas. Em caso de desacordo nas respostas destas réplicas (ocorrência de falhas) durante o processamento de uma requisição, f novas réplicas são ativadas no sentido de obter um quórum mínimo de $2f+1$, o que permite o acordo sobre a resposta de no mínimo $f+1$ réplicas corretas.

Como queremos desempenho e tratar somente com intrusões (não tratamos com faltas de *crash*), o nosso modelo *IT-VM* se mostrou como uma base adequada para o uso em *IdPs*. O *IT-VM* é apresentado em seus elementos no contexto de um *IdP* tolerante a intrusões na sequência deste capítulo.

4.4 PROVEDOR DE IDENTIDADES TOLERANTE A INTRUSÕES (*IT-IDP*)

O modelo de um Provedor de Identidades tolerante a intrusões (*IT-IdP*) é apresentado nesta seção, fazendo o uso de virtualização no sentido de manter seu comportamento correto mesmo diante de ataques e falhas. A Figura 12 apresentada este *IT-IdP* com a estratificação de suas funcionalidades em camadas.

O esquema do *IT-VM* se faz presente neste modelo com seus elementos: as réplicas isoladas em suas *VMs* e, mantidos isolados em zona confiável pelo *Hypervisor* da tecnologia de virtualização, as abstrações de *Agreement Service* e de memória compartilhada. A estes elementos mantidos na zona confiável é ainda acrescentado o Serviço de Autenticação, caracterizando então o *IT-IdP* mostrado na figura citada.

No nível mais alto do modelo da Figura 12, *IdP proxies* são dispostos cada um em suas próprias máquinas virtuais (*VMs*), correspondendo então à replicação do esquema *IT-VM*. Estes *proxies* são réplicas executando os protocolos da especificação *OpenID*, sendo os únicos elementos do modelo visíveis via rede través de diferentes portas e endereços, portanto, sujeitos a ações de atacantes. Os *IdP proxies* na verdade não controlam registros de usuários, mas passam as informações de *login* obtidas via protocolos *OpenID* para o *Agreement Service* que é mantido isolado do ambiente de rede. Esta é a razão por que estes *IdPs* na camada mais alta, que executam os protocolos do *OpenID*, são chamados de *proxies*.

Os *IdPs* proxies do *OpenID*, executando em suas *VMs*, por atuarem como provedores com diferentes endereços, são então vistos externamente como diferentes provedores de identidades *OpenID*. Seus comportamentos são considerados como corretos ou falhos em uma requisição de *login* de usuário. *IdPs proxies* (e suas máquinas virtuais) são ditos corretos se mantêm a execução do protocolo *OpenID* seguindo suas especificações, enquanto *proxies* maliciosos ou falhos podem apresentar qualquer comportamento arbitrário, tal como: adulterar mensagens; omitir seus recebimentos; omitir envios de asserções e certificar indivíduos não autorizados (produzindo asserções falsas). A replicação em nosso modelo aplicada através dos *proxies OpenID* implica em uma replicação sem estado. Ou seja, não necessitamos das usuais técnicas de replicação ativa (replicação Máquina de Estados (SCHNEIDER, 1990)) para a tolerância a intrusões nos protocolos. Esta situação simplifica em muito a manutenção e gerenciamento das réplicas *proxies*.

Ainda no nível mais alto desta figura, está o Repositório de Descritores (*RD*) que executa também sobre uma máquina virtual. O repositório *RD* mantém informações sobre os *OpenID proxies* da configuração atual. Estas informações em *XRDS* (OASIS, 2008) são acessadas pelos provedores de serviço via protocolo *Yadis* (MILLER, 2006), permitindo então a descoberta destes *IdPs proxies*.

No nível subsequente da Figura 12, abaixo dos *IdPs proxies* e do *RD* com suas respectivas *VMs*, está a zona confiável do modelo constituída pelo monitor de máquinas virtuais (*VMM*) onde são implementadas as abstrações de *Memória Compartilhada* e do *Agreement Service* (*AgS*). Estes componentes da zona confiável não são visíveis via rede. Na verdade, as intrusões devem se limitar às *VMs* dos *IdPs proxies*, que são os únicos componentes do modelo visíveis externamente.

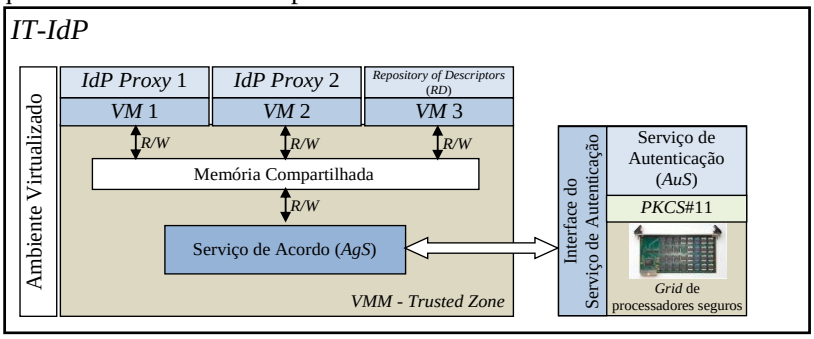


Figura 12 - *IdP* Tolerante a Intrusões

A Memória Compartilhada é usada para a comunicação dos *IdPs Proxies* com o *Agreement Service*. O *AgS* desempenha funções vitais da arquitetura da Figura 12. Entre suas funções estão: a verificação das requisições *OpenID* de *login* recebidas dos clientes através dos *IdPs proxies*; a comunicação com o Serviço de Autenticação (*AuS*); e a criação de asserções *OpenID* que devem refletir o resultado do *login*. Estas asserções são mantidas no *AgS* durante as sessões de computação dos usuários e cópias são liberadas sob demanda de provedores de serviços (*SPs*) seguindo os protocolos do *OpenID*.

É bom ressaltar também que o *Agreement Service* é o responsável pelo gerenciamento das máquinas virtuais dos *IdPs proxies*. Ou seja, é o *AgS* que mantém a configuração de duas *VMs* com os respectivos *OpenID proxies*. E quando um comportamento malicioso é detectado, o *AgS* então retira a réplica corrompida e ativa uma nova *VM*, compondo então o quórum de execução normal de duas *proxies* e suas *VMs*. A garantia de correção do *Agreement Service* é baseada em premissas de isolamento em

relação à rede externa e às *VMs* dos *proxies*. A sua simplicidade torna fácil a verificação e testes sobre o código do mesmo.

O *Serviço de Autenticação* (*AuS*) é o componente que realmente faz as validações de credenciais de usuários durante os processos de *login*. Além disto, o *AuS* é também envolvido com as verificações e produção de assinaturas necessárias no protocolo de *login* e ainda, com a liberação de atributos de usuários para os *SPs* solicitantes, conforme as especificações *OpenID*. O Serviço de Autenticação que mantém credenciais e atributos de usuários, foi construído sobre *hardware* separado e especializado no projeto *SecFuNet*.

Este serviço de autenticação é isolado da rede externa e dos níveis onde o nosso modelo de tolerância a intrusões atua. As interações com o *AuS* se dão via o *Agreement Service* que é a única entidade interagindo com o mesmo através de canal isolado e usando uma interface (interface *Serviço de Autenticação*).

Esta interface oferece a operação *verifyUserCredentials*() que permite a verificação de *userids* e credenciais (de usuários) passados como parâmetros nas ativações da mesma. O resultado no retorno desta operação permite ao Serviço de *Agreement* (*AgS*) montar a asserção de autenticação correspondente. A operação *createSign*() é também parte desta interface do Serviço de Autenticação (*AuS*) e deve retornar uma assinatura feita com a chave privada do *AuS* sobre os dados passados como parâmetros de entrada. O *AgS* ativa sempre esta última operação quando precisa de assinaturas nas asserções de autenticação e de atributos. O isolamento do *AuS* de acessos via redes externas e das próprias *VMs* locais, também é necessário para garantir a correção e a invulnerabilidade do modelo.

O *login* de um usuário é realizado através de *userid*/certificados, sendo que estes certificados podem ser obtidos de uma *PKI* oficial ou formados a partir dos identificadores e chaves públicas de usuários assinados pelo serviço de autenticação de seus domínios (com o uso da chave privada do *AuS* do *IT-IdP* do domínio do usuário). No projeto *SecFuNet*, as interações de *login* de nossos protótipos faziam a liberação do certificado e de "prova de atualidade" diretamente do *smartcard* do usuário. Esta prova de atualidade é obtida com assinatura sobre o *userid* concatenado com a data do acesso, usando a chave privada do usuário presente no *smartcard*.

As nossas premissas de correção do modelo são fundamentadas no isolamento conseguido através do Monitor de Máquina Virtual (*VMM - Virtual Machine Monitor*) da tecnologia de virtualização que pode apresentar vulnerabilidades, porém estas não podem ser exploradas

externamente via rede ou pelas máquinas virtuais locais. Ou seja, o *AuS* e o *AgS* não são comprometidos através de intrusões via rede.

4.5 INTEGRAÇÃO DO *IDP* TOLERANTE A INTRUSÕES COM PROTOCOLOS *OPENID*

O *OpenID* é um *framework* para gerenciamento de identidades (seguindo o modelo centralizado) que usa protocolos e padrões conhecidos (HTTP (R. FIELDING et al., 1999), HMAC (H. KRAWCZYK; BELLARE; CANETTI, 1997), "*Diffie-Hellman Key Agreement Method*" (E. RESCORLA, 1999), SHA1(D. EASTLAKE; JONES, 2001), entre outros. Neste *framework*, o protocolo de identificação inicia com um usuário se apresentando a um provedor de serviços (*SP*) através de um identificador *OpenID*[7]. Este identificador é usado em provedores de serviço na localização do *IdP OpenID* gerador deste identificador. É necessário que o *SP* que fez a "descoberta" deste *IdP* tenha relações de confiança com o mesmo para delegar a autenticação do usuário indicado pelo identificador *OpenID* correspondente.

O protocolo de autenticação *OpenID* é representado em seus passos na Figura 13. Este protocolo inicia com o usuário, através de seu navegador (*user-agent*), acessando o provedor de serviços (*SP*) (Passo1). O *SP* então apresenta no navegador do usuário um formulário solicitando o identificador *OpenID* do cliente (Passo2) que, na sequência, no passo 3, apresenta o seu identificador de usuário (identificador *OpenID*).

Depois é feita a normalização (BERNERS-LEE; FIELDING; MASINTER, 2005) deste identificador que tem por objetivo permitir então a descoberta do *IdP* para autenticar o usuário (*Discovery*) (Passo 4). A descoberta é um processo executado pelo *SP* com o objetivo de obter informações necessárias deste provedor de identidades (*IdP OpenID*) para o início das trocas de mensagem com o mesmo. Esta descoberta pode ser realizada usando documentos *XRDS* ou através de uma *URL*. A obtenção de documentos *XRDS* contendo os endereços dos *IdPs* que podem autenticar este usuário é feita usando o protocolo *Yadis* (MILLER, 2006), através de identificadores *XRI* (*Extensible Resource Identifier* (OASIS, 2008)). Quando a descoberta é feita com uma *URL*, então métodos HTTP são usados para a localização do *IdP*. Em ambos casos, tanto o identificador *XRI* como a *URL* devem ser obtidos diretamente do identificador *OpenID* do usuário.

---

[7] Nas especificações *OpenID*), o provedor de serviços (*SP*) é identificado como *Relying Party* (*RP*). Neste capítulo mantemos a notação clássica de *SP*.

De posse das informações necessárias, o *SP* pode então estabelecer associação direta ou indireta com o *IdP* (Passo 5). A associação direta é opcional e usa troca de segredos entre as partes segundo o protocolo *Diffie-Hellman Key Agreement*. O canal direto de comunicação tem as trocas feitas usando HTTP POST. A alternativa mais usada é o encaminhamento de trocas entre o *SP* e o *IdP* via HTTP *redirects* provocando comunicações indiretas via *browser* do usuário.

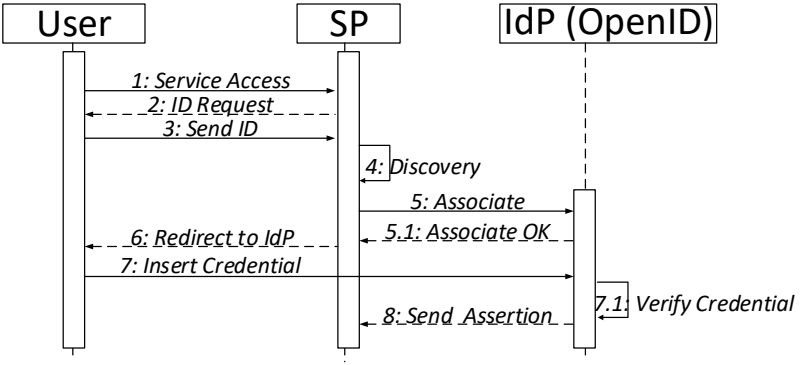


Figura 13 – Protocolo OpenID

No Passo 6, o *browser* do usuário é redirecionado para o *IdP OpenID* de modo que esse faça o seu *login*, inserindo suas credenciais de acesso (Passo 7). O Passo 8 descreve o envio ao *SP* pelo *IdP* da asserção de autenticação, com as informações sobre o processo de *login* correspondente. Esta asserção pode ser positiva quando o *IdP* reconhece o usuário e sua autenticação ou, em caso contrário, negativa. Um dos campos importantes da asserção é a assinatura do *IdP* que garante a legitimidade do processo para o *SP*. Caso o usuário já tenha se identificado junto ao seu *IdP*, o *SP* (provedor de serviço) simplesmente solicita diretamente ao *IdP* a asserção de autenticação correspondente. No final da identificação positiva, o usuário tem o seu *browser* redirecionado para o *SP*, para solicitar os seus acessos aos recursos, seguindo as políticas de autorização do *SP*.

A integração do modelo *IT-IdP* e de suas entidades é feita de modo a não resultar em modificações nos passos usuais do protocolo do *OpenID* e de deixar transparente a replicação usada neste *IdP*, tanto para usuários como para provedores de serviços (*SPs*). A Figura 14 ilustra os passos do protocolo *OpenID* adaptado para a tolerância a intrusões do modelo *IT- IdP*.

Este protocolo estendido funciona de maneira similar ao protocolo *OpenID* descrito acima até o passo 4. Como a abordagem de

replicação do *IT-IdP* conta sempre com mais de um servidor (são dois *OpenID proxies*), o passo de descoberta deve ser feito obrigatoriamente utilizando o protocolo *Yadis* que retorna para o *SP* os descritores (*Yadis Resource Descriptor*) indicando os provedores de identidade que podem ser usados para autenticar o usuário[8]. Ou seja, são retornados ao *SP* os descritores dos dois *IdP OpenID proxies* presentes no modelo *de* replicação. Estes descritores definem também a prioridade de uso destes *OpenID proxies*. No nosso modelo, estas informações liberadas pelo *Yadis* estão disponíveis no Repositório de Descritores (*RD*), em máquina virtual visível via rede.

Para não modificar o protocolo *OpenID* nas interações entre *browser*, *SP* e *IdPs*, foi necessário adaptar o modelo *IT-VM* para replicação passiva, pois este inicialmente previa replicação ativa. Na lista de descritores retornada ao *SP* na descoberta com o *Yadis*, os diferentes *proxies* da replicação de *IdPs* são indicados pelas suas prioridades aos seus diferentes papéis (primário e *backup*) na replicação passiva. Nesta nova situação, o *IdP proxy* no papel de "primário" deve interagir com o *browser* e o *SP*. Com isto, o *SP* (provedor de serviços) deve estabelecer a associação do Passo 5 com o *proxy* primário. Mas, caso este *proxy* apresente comportamento malicioso, a outra réplica ativa da configuração (o *IdP proxy* que atua como *backup*) passa a assumir a função de novo primário e é usada então pelo SP nas suas interações com o *IT-IdP*.

No passo 5, o *SP* contata o *proxy* primário para verificar se o usuário se encontra autenticado (ou seja, se existe asserção ainda válida no *IT-IdP*). Esta consulta é verificada junto ao *Agreement Service* que mantém cópias de asserções de autenticação de usuários com sessões de computação ativas. Em caso negativo, o *browser* do usuário é redirecionado para o *proxy* primário (passo 6). O *proxy* primário recebe então as credenciais do usuário em mensagem *OpenID* (HTTP POST com os dados do formulário disponível no *browser*, no Passo 7), extrai então estas credenciais e escreve as mesmas na memória compartilhada para que sejam lidas pelo *Agreement Service* (passo 8).

[8] O usuário apresentando o seu identificador *OpenID* ao provedor de serviços, no processo chamado de normalização, este *SP* obtém deste identificador o local onde está hospedado o documento (*XRDS Document*) com os descritores para *IdPs proxies*.

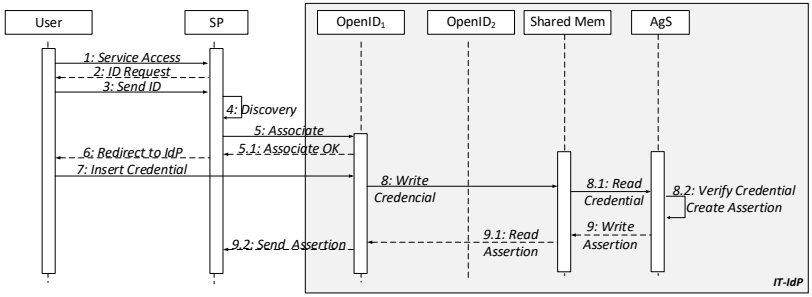


Figura 14 - Protocolo OpenID estendido para a tolerância a intrusões

O *Agreement Service* deve então submeter as credenciais do usuário às verificações correspondentes, usando a operação *verifyUserCredentials*() da interface do Serviço de Autenticação (*AuS* na Figura 12). Se a operação de verificação de credenciais for validada o *Agreement Service* pode criar a asserção de autenticação correspondente ao *login* do usuário. Esta asserção tem informações como a identidade do usuário, a *URL* apontando para o *IdP proxy* no papel de primário, um *handle* da associação entre *SP* e o *proxy* primário, o *nonce* e ainda a assinatura do *AuS* sobre todos estes campos. A assinatura destas informações é conseguida pelo *Agreement Service* (*AgS*) usando a operação *createSign*() da interface do Serviço de Autenticação. Para completar o passo 9 (Figura 14), o *Agreement Service* deve passar esta asserção para o *proxy* primário para que este, por sua vez, envie através do protocolo *OpenID* esta asserção ao provedor de serviço (*SP*).

As verificações de autorização, nas demandas por recursos, são baseadas nestas asserções de autenticação recebidas de *IT-IdPs* da federação. Com isto, as *clouds* devem manter as listas dos *IdPs* com os quais possuem relações de confiança e seus respectivos certificados (chaves públicas dos *AuS* destes *IT-IdPs*). Baseadas nestas listas, estas *clouds* podem validar as asserções de autenticação emitidas pelos *AgS* (*Agreement Services* dos *IT-IdPs*) da federação. Feitas as verificações sobre uma asserção por um *SP*, o acesso requisitado pelo usuário ainda pode ficar sujeito a verificações de políticas de autorização locais. Vale a pena ressaltar que no protocolo do *IT-IdP*, mesmo envolvendo a inclusão de replicação (os *IdP proxies*), não foram necessárias modificações em mensagens *OpenID*.

4.6 EXECUÇÕES ANORMAIS DO PROTOCOLO IT-IDP

Apesar de todo o isolamento das entidades críticas, ataques aos *IdPs proxies* não podem ser evitados. Na abordagem proposta, quando as réplicas *proxies* são comprometidas por intrusões, não há uma grande preocupação quanto a possíveis modificações de asserções ou mesmo o

envio de credenciais falsas, uma vez que estas informações estão sujeitas a assinaturas ou a verificações envolvendo as partes confiáveis do *IT-IdP* que são o *Agreement Service* e o Serviço de Autenticação (isolados e livres de intrusões).

Atacantes podem corromper réplicas de *IdP proxies* de modo que estas não executem corretamente, segundo as especificações, o protocolo *OpenID* e as trocas com os componentes confiáveis. O comportamento não correto na execução dos protocolos pode ser expresso no fato que, uma réplica comprometida (*IdP proxy*) pode se recusar a transferir informações entre o usuário, o *SP* e o serviços da zona confiável do *IT-IdP* (*AgS* e o *AuS*). Por exemplo, a réplica primária sendo maliciosa, pode não informar as credenciais do usuário ao *AgS* e *AuS* ou ainda, não enviar a asserção de autenticação que disponha ao *SP* solicitante. Os mecanismos usados em conjunto com a replicação de *proxies* devem detectar estas possíveis ações e remover o *proxy* malicioso do *IT-IdP*.

A Figura 15 ilustra o comportamento do protocolo *IT-IdP* em presença de malícia. Os passos 1 a 7 ocorrem da mesma forma como apresentado na seção anterior. Lembrando que no passo 4, onde é executado o protocolo de descoberta *Yadis*, é retornada para o *SP* a lista de *proxies* do *IT-IdP* que este pode fazer uso. Porém, na instância de protocolo apresentada na Figura 15, é mostrado o comportamento errôneo do *proxy* primário da replicação ($OpenID_1$) que se recusa a inserir as credenciais enviadas pelo usuário na memória compartilhada (Passo 8), impedindo que o protocolo possa prosseguir na sua execução normal. A detecção desta anomalia é conseguida com *timeout* no *SP* que solicitou a autenticação do usuário. Dessa forma, o *SP* ativa seu temporizador, aguardando uma asserção de autenticação como resposta, quando faz a demanda de autenticação no Passo 6. Se estas não chegam dentro de prazo determinado, o *SP* utiliza-se da lista de descritores retornados pelo *Yadis*, para tentar conexão com outro *proxy* do *IT-IdP* (a réplica *backup*). No passo 9, com o temporizador do *SP* excedido, este redireciona o pedido de autenticação do usuário para o *proxy backup* ($OpenID_2$) e a execução do protocolo prossegue com o redirecionamento do usuário para o *proxy OpenID2* no sentido que apresente suas credenciais (passo 10). O usuário então apresenta seu certificado ao novo primário ($OpenID_2$). Todas as situações de comportamentos maliciosos já citados são detectadas a partir de temporizações no provedor de serviços (*SP*).

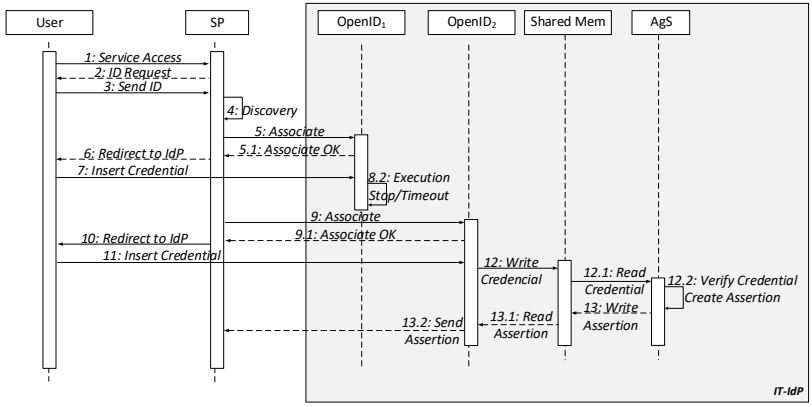


Figura 15 – Protocolo *OpenID* replicado na presença de intrusões

4.7 PROCESSO DE LOGOUT NO MODELO *IT-IDP*

A especificação *OpenID* define duas formas de *logout*: a primeira, partindo do *RP/SP* para o *IdP*, onde o *RP/SP* solicita que a sessão de autenticação do usuário seja terminada. Para isso, o *RP/SP* fornece um *endpoint* de *logout*. A segunda, quando o *IdP* é quem define a sinalização de *logout* de um usuário. Este mecanismo envolve trocas mais complexas, uma vez que todos os *RPs/SPs* devem ser notificados desta ação. De acordo com as especificações, os *IdPs OpenID* mantêm uma lista de "sites visitados", os quais mantém *back-channels* para o acesso a esta funcionalidade dos *RPs/SPs*.

Ao receberem uma requisição de *logout*, os *RPs/SPs* devem fazer diversas validações, tais como, verificação de identificação do *IdP*, de identidade do usuário autenticado e principalmente da assinatura da requisição. Esta assinatura deve estar baseada nas chaves criadas no processo de associação efetuado nas trocas iniciais de *login* de usuário.

Em nosso caso, como removemos as diversas funcionalidades do *IdP*, qualquer um dos processos de *logout* deve, obrigatoriamente, notificar o *AgS* para que este remova a asserção do usuário gerada no processo de *login*. Para isso, as mensagens de *logout* geradas pelo protocolo *OpenID* são encaminhadas para o *AgS* através da memória compartilhada, assim como as demais mensagens do protocolo.

Como o processo de *logout* descrito nas especificações do *OpenID* definem somente mensagens de *logout* que não precisam necessariamente de uma mensagem de retorno ou resposta. Neste sentido, uma simples notificação para o *AgS* é suficiente para que esta funcionalidade seja implementada em conformidade com as

especificações, mesmo com o modelo IT-*IDP* apresentando modificação estrutural significativa se comparado à especificação original do *OpenID*.

4.8 CONSIDERAÇÕES SOBRE O MODELO *IT-IDP*

Sempre que um *SP* se conecta com uma associação à réplica *backup*, o *proxy* primário é assumido como malicioso e é removido junto com sua máquina virtual (*VM*), sendo em seguida substituído na replicação por uma nova réplica *proxy* que fará o papel de *backup*. Esta nova réplica terá endereços IP e porta diferentes do *proxy* removido e, portanto, o arquivo *XRDS* mantido no Repositório de Descritores deve ser atualizado com informações da nova réplica. O estabelecimento de associação entre o *SP* e os *OpenID proxies* faz uso das partes confiáveis do *IT-IdP*, principalmente do suporte criptográfico do Serviço de Autenticação (*AuS*). O *Agreement Service* ao receber informações do estabelecimento de uma associação com um *proxy backup* dá prosseguimento normal ao protocolo até o passo 13 da Figura 15 e, de forma concorrente, remove a antiga primária que é assumida como maliciosa, ativando a nova *backup* e promovendo as alterações necessárias em documento *XRDS* que envia de imediato através da memória compartilhada da zona confiável para o *RD*.

Com a remoção e a imediata ativação de uma nova *VM*, a configuração das réplicas-*proxies* é sempre restaurada, de modo que o *IT-IdP* mantém sempre no seu ciclo de vida o mesmo número de réplicas. Estas substituições são muito simples porque os *proxies* não mantêm informações de estado. As informações de estado (informações de conexões, asserções e demandas de autenticação) estão contidas no *Agreement Service* que é mantido protegido, isolado da rede externa.

Estas reconfigurações geram algumas implicações: o *SP* pode também não ter resposta da nova primária e possui uma cópia do documento *Yadis* (arquivo *XRDS*) ainda com a configuração anterior do *IT-IdP*. Com a sinalização de seu temporizador e o esgotamento das possibilidades do arquivo *XRDS* que possui, o *SP* inicia todo o processo de descoberta novamente, recorrendo ao repositório de descritores para obter nova configuração. Ou seja, no documento *XRDS*, descritores devem indicar os novos *proxies*, com suas prioridades definido seus papéis nesta nova configuração. O *SP* então pode recomeçar novamente o contato com *proxies* da configuração do novo arquivo. O *Agreement Service* como agente destas reconfigurações, deverá sempre manter através do *RD* o arquivo *XRDS* com a configuração mais recente do *IT-IdP*.

Temos que inferir que os provedores de *clouds* de uma federação (os *SPs*) são confiáveis, uma vez que estes devem delegar todas as suas

autenticações de usuários para uma entidade externa (o *IT-IdP*). A confiança mútua deve ter sido construída através de contratos. Meta dados foram trocados estabelecendo esta confiança em seus sistemas computacionais (por exemplo, estabelecimento de políticas mútuas, troca de certificados, definição de entidade certificadora, etc.) o que impede que comportamentos indesejados de um *SP* não sejam detectados.

Resta então assumir como atacante, indivíduos externos a estas relações de federação. Um atacante que consiga inferir a URL a partir do identificador *OpenID* de usuário, pode se utilizar do protocolo *Yadis* e ter acesso ao arquivo *XRDS* no *RD*. E de posse deste documento, este atacante poderia promover ataques aos proxies de modo a provocar reconfigurações sucessivas e intermináveis do *IT-IdP*. Para evitar estes acessos e ataques indesejáveis, os arquivos *XRDS* são então criptografados com chaves de criptografia assimétrica. Estas chaves assimétricas usadas nestes arquivos são protegidas com encriptações, concretizadas com o uso das chaves públicas de todos os *SPs* associados do *IT-IdP*[9]. Estas encriptações são mantidas anexadas ao mesmo arquivo *XRDS* de modo a que todos os *SPs* com relações de confiança com o *IT-IdP* possam ter acesso a chave usada na proteção do arquivo *XRDS* e ao conteúdo de suas informações. Isto é factível, mesmo em situação de trocas periódicas desta chave porque em qualquer situação um *IdP* não possui mais do que algumas poucas dezenas de *SPs* associados.

No nosso modelo, o repositório de descritores (*RD*), é um serviço provido através de uma *VM*, mantendo também endereço e porta próprios, e poderia sofrer também ataques de *DoS* (*Denial of Service*), tentando deixá-lo indisponível. Nesta situação, como os *SPs* em qualquer situação de federação não representam em números grande quantidade, atualizações do arquivo *XRDS* poderiam ser enviadas como mensagens diretamente aos *SPs* e tratadas por estes últimos como meta dados. Isto reduziria o passo de descoberta dos protocolos apresentados anteriormente e afastariam ataques sobre o *RD*. Na nossa versão atual, mantemos o *RD* como repositório destes documentos *XRDS*, deixando as nossas trocas ainda compatíveis com as especificações *OpenID*. Mas, mudanças para mensagens de atualização enviadas a partir do *IT-IdP* não representam alterações importantes nas nossas implementações.

Em situações totalmente ao acaso, *proxies* primários poderiam sofrer ataques de *DoS*, mas com a ação de um *SP* confiável acionando o *backup*, e contando com o escalonamento *fair* do suporte de máquinas virtuais (todas as *VMs* possuem a mesma prioridade junto ao

[9] Os certificados dos *SPs* associados são mantidos pelo *AuS*.

escalonador), então logo o primário sobre ataque estaria fora da configuração e os atacantes perderiam o endereço para continuar suas ações de ataque.

Uma possível fragilidade no sistema é o caso do usuário apresentar as credenciais corretas e o provedor de identidades comprometido alterar estas credenciais ao escrevê-las na memória compartilhada, impedindo que um usuário com credenciais corretas possa se autenticar no sistema. No intuito de aumentar a robustez da nossa solução, um sistema envolvendo *smartcards* como fonte de credenciais foi também desenvolvido.

A substituição do mecanismo de autenticação para o uso de *Smartcards* aumenta o nível de segurança na medida em que, para se autenticar, não basta conhecer o segredo do usuário, mas também ter posse do dispositivo físico. Cada *Smartcard* contém um par de chaves e um certificado associado, que somente podem ser utilizados mediante apresentação do PIN secreto. Toda a criptografia é executada pelo processador do próprio *Smartcard*, de forma que a chave privada nunca fica exposta. Apesar do PIN se assemelhar a uma senha, o valor secreto é verificado localmente na máquina do usuário e somente tem algum valor se aliado à posse do cartão físico. Através do uso das chaves mantidas pelo *Smartcard*, as credenciais do usuário são criptografadas e assinadas de forma que qualquer alteração efetuada por um *IdP* malicioso seja detectada pelo *AgS*. Mais detalhes da implementação deste modelo envolvendo *Smartcards* são apresentados em (BOGER et al., 2014).

4.9 IMPLEMENTAÇÃO DE PROTÓTIPO E ANÁLISE DE RESULTADOS

De forma geral, a estrutura utilizada em nosso protótipo é representada pela Figura 16. Para a replicação dos *IdP proxies* foi utilizado o software de virtualização XEN, para a execução das máquinas virtuais. Além disso, o uso do *hypervisor* XEN permitiu a implementação da memória compartilhada para a comunicação entre as VMs *proxies* e o *Agreement Service*. A memória compartilhada e o *Agreement Service* são implementados no nível do *hypervisor* (no domínio 0), o que permitiu o isolamento destas abstrações, de forma que ambas possam ser consideradas componentes confiáveis, que é premissa básica da proposta. Nas VMs de proxies foram utilizadas diferentes versões do sistema operacional Linux, com o objetivo de manter certa independência de vulnerabilidades já que estas máquinas são os únicos componentes do *IT-IdP* visíveis externamente (via rede). Na VM *Repository of Descriptors*

é disponibilizado o documento *XRDS* contendo os descritores para os *IdPs proxies*.

Para provedor de identidades (*OpenID*) foi utilizada a implementação em código livre *Java OpenID Server* (JOIDS, 2009), que foi alterada para que o protocolo fosse executado diante das características de replicação e uso de componentes confiáveis, como descritas nas seções anteriores. Vale salientar que foram removidos da implementação *JOID* os módulos de gerenciamento de conta de usuários. No nosso modelo, estas contas são gerenciadas de fato pelo Serviço de Autenticação (*AuS*). O provedor de identidades *OpenID* implementado é executado sobre o servidor de aplicação Apache *Tomcat*.

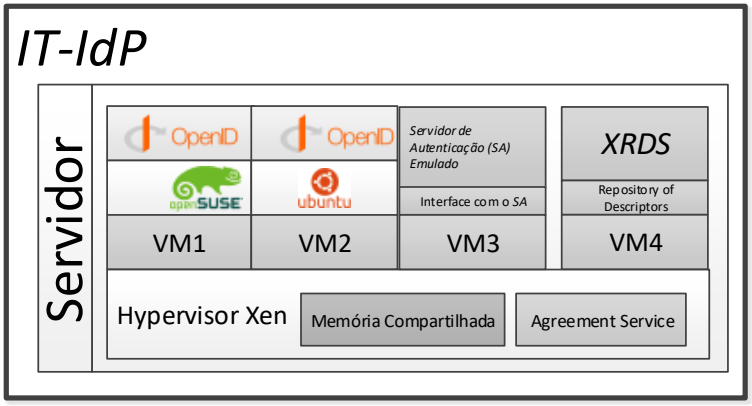


Figura 16 – Estrutura do sistema

O provedor de serviços (*SP*) desenvolvido não precisou de extensões especiais em relação ao protocolo *OpenID*, mas este tem que saber lidar com a indisponibilidade momentânea de um *IdP proxy*. Diante disto, este *SP* deve contatar outro *proxy* (como alternativa do *IT-IdP*). Fizemos uso da tecnologia AJAX (*Asynchronous Javascript and XML*(W3.ORG, 2012)) para o desenvolvimento do provedor de serviços. O uso do AJAX nos permitiu que houvesse um maior controle sobre a aplicação e suas trocas de mensagens com o *IT-IdP*. Ainda que ocorram trocas de provedor de identidades (na verdade trocas de *IdPs proxies*), toda a interação é feita pela aplicação AJAX executada no navegador do cliente através dos códigos *JavaScript* do *engine* utilizado. O usuário não precisa começar todas as trocas de novo a partir do *SP*. Na substituição de *IdP* primário, a página de *login* do *OpenID* é reativada novamente no *browser* do usuário.

O *Agreement Service* e outras entidades introduzidas pelo modelo *IT-IdP* foram desenvolvidos com a linguagem de programação Java. O

servidor de autenticação, mantido completamente isolado da rede, é quem realmente gerencia as contas de usuários e foi desenvolvido no *SecFuNet* usando um *grid* de microcontroladores seguros[10], cada um desses com *software* embutido. A estrutura geral do servidor de autenticação é concretizada sob um sistema operacional dedicado. Como o foco desta seção é o ambiente virtualizado que deve tolerar intrusões e interagir com o servidor de autenticação, emulamos o comportamento deste servidor a partir de uma máquina virtual isolada.

4.9.1 Experimentação do Protótipo

Com o objetivo de testar nossas proposições o protótipo foi executado em um servidor com processador Intel Core i7 3.4 *Ghz*, com 8 GB de memória conectado com seus clientes através de uma rede local Ethernet funcionando a 100 *Mbps*.

Como a utilização de provedores de identidade tem grande dependência de interações com seus usuários (no fornecimento de credenciais e/ou atributos), em nosso teste usamos o envio automático de dados de identificação a partir de uma máquina cliente. A arquitetura dispõe de duas réplicas *proxies* executando o provedor de identidades *OpenID* e, imediatamente a detecção, toda réplica detectada como corrompida é removida e outra é ativada em seu lugar. Os reflexos no desempenho não são muito sentidos porque a iniciação de novos *proxies* não envolve transferências de estado.

Realizamos testes envolvendo três rodadas de 1000 operações de autenticação. Estes testes foram divididos em duas classes: i) execução de autenticações sem falhas maliciosas ii) execução de autenticações com falhas maliciosas nos *proxies*. Na primeira classe de testes as simulações não levaram em conta a presença de intrusões no sistema (testes sem falhas), portanto, com o serviço funcionando sempre sem a necessidade da troca do *proxy* primário.

Na segunda classe, foram incluídas modificações no sistema de forma que intrusões fossem simuladas. Num primeiro momento fizemos simulações com faltas (maliciosas) no domínio de valores. Ou seja, os proxies maliciosos quando alteravam informações de credenciais de usuários eram detectados pelos *AgS e AuS* do *IT-IdP* e, em situação inversa quando faziam ataques à integridade das asserções emitidas pelo *AgS*, eram detectados pelo *SP*. Em ambas situações de teste, o *proxy* primário foi sempre removido.

---

[10] As placas com o *grid* de microcontroladores seguros são um produto de propriedade intelectual das empresas Implementa, EtherTrust and Telecom ParisTech.

Basicamente os resultados que merecem uma atenção especial são as falhas no domínio do tempo. O não repasse de informações pelos proxies são importantes e são detectados com *timeout*. As simulações de faltas no domínio do tempo foram feitas em uma rede local, onde os tempos de respostas são rápidos. Como resultado, nestes testes, foram calculadas as médias dos tempos de resposta nos processos de identificação.

Como em nossa abordagem a replicação é renovada, montamos casos de testes com diferentes distribuições de réplicas maliciosas. As distribuições são apresentadas na Tabela 7. A coluna $\boldsymbol{f}_{\%}$ apresenta, dentro do conjunto de 1000 requisições, qual a porcentagem de falhas maliciosas que ocorrem no conjunto. A coluna $\boldsymbol{N_f}$ apresenta o número destas falhas e a coluna $\Delta\boldsymbol{r}$ identifica os intervalos com autenticações corretas (ou entre autenticações com falhas). Em alguns dos nossos experimentos, optamos pela ocorrência de falhas maliciosas em determinadas situações de autenticação para que os custos das recuperações correspondentes se tornassem mais visíveis.

Na Figura 17 é apresentado o tempo médio de autenticação nas duas classes de testes citadas acima. No caso de teste onde não existem *proxies* maliciosos, o tempo médio foi de 7 *ms*. Este valor de tempo é calculado considerando o intervalo entre o envio de credenciais pelo *browser* e a recepção de asserção correspondente no *SP*.

Nos demais testes, onde as intrusões foram simuladas, este tempo foi crescente em relação ao aumento do número de intrusões. Este crescimento está relacionado ao tempo de renovação da replicação no algoritmo. No nosso esquema de testes, mantemos duas réplicas participando da configuração de *IdP* proxies, conforme definido no modelo do *IT-IdP*. Mas, para evitar os custos de criação de novas réplicas (em torno 1 segundo), foram deixadas oito réplicas (*VMs* com os seus proxies) em espera mas fora da configuração. A remoção de uma faltosa envolve a ativação destas réplicas em espera. Vale a pena salientar que o teste com 100% de falhas reflete que, a cada requisição de autenticação ocorra uma intrusão. Nesta situação, pela Figura 17, chegamos à conclusão que uma autenticação onde ocorre uma intrusão, o tempo médio para esta autenticação ocorrer como esperado é de 1,084 segundos.

Apesar de não haverem comparações com outras abordagens de forma a atestar a eficiência do algoritmo proposto, os testes serviram para demonstrar que a abordagem é factível e como esta se comporta em um ambiente hostil onde intrusões são frequentes. A comparação com outras abordagens também não foi efetuada por não ter sido encontrado na literatura trabalhos sobre provedores de identidade tolerantes a intrusão.

Tabela 7 - Distribuição das faltas

$f_{\%}$	N_f	Δr
0,1%	1	*
1,00%	10	99
2,50%	25	39
5,00%	50	19
10,00%	100	9
12,50%	125	7
16,60%	166	5
25,00%	250	3
50,00%	500	1
100,00%	1000	0

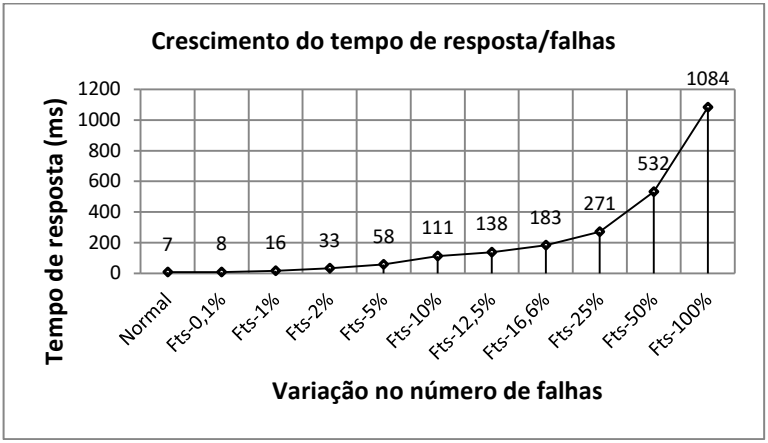


Figura 17 – Tempo de resposta dos testes

4.10 TRABALHOS RELACIONADOS

Nesta seção, alguns trabalhos citados são confrontados ou mesmo usados como mecanismo de apoio no sentido de evidenciar resultados de nossa proposta. Mas é bom evidenciar que até o momento não temos conhecimento de experiência similar a nossa. Ou seja, não foram encontradas na literatura abordagens que tratassem de tolerância a intrusões em provedores de identidades. Porém, o uso de provedores de identidade vem sendo crescente em aplicações e sistemas distribuídos de larga escala como computação em nuvem, organizações virtuais, computação em grade, etc.

Em (COPPOLA et al., 2012) é descrito uma infraestrutura para federação de provedores de computação em nuvem chamada *Contrail*. Esta infraestrutura é responsável pelo gerenciamento dos recursos

oferecidos por nuvens federadas. Dessa forma, quando o cliente deseja implantar sua aplicação em nuvem, o *framework* citado seleciona a nuvem que melhor possa atender aos requisitos de *SLA* (*Service Level Agreement*) negociados com o cliente. Do ponto de vista do gerenciamento de identidades, ainda que as aplicações de um cliente estejam sendo executadas em várias nuvens distintas, este não deve necessariamente se apresentar com uma identidade diferente em cada um dos provedores das mesmas. Desta forma, na abordagem *Contrail* é definido um nível de gerenciamento de identidades que estabelece relações de confiança entre os provedores de identidade usados nas diferentes nuvens. Estas relações de confiança definem então uma federação de nuvens. Para tanto, foi utilizado nesta infraestrutura o protocolo *OAuth2*.0 (MICROSOFT, 2012) para a geração de *tokens* de autenticação e de autorização e documentos SAML (*Security Assertion Markup Language*) para troca de certificados entre os *IdPs* de diferentes nuvens. Os *IdPs* desta proposta são implementados nas próprias nuvens, o que certamente pode ser um obstáculo significativo para se garantir a segurança dos mesmos. Dentre os trabalhos futuros propostos pelos autores está a integração do protocolo *Shibboleth* para identificação dos usuários.

Outro trabalho que propõe o uso de federação a partir de *IdPs* de diferentes nuvens é apresentado em (TUSA et al., 2012). Neste trabalho são discutidos principalmente detalhes da implementação de autenticação única (*Single Sign On*) através da integração do protocolo *Shibboleth*, SASL (*Simple Authentication and Security Layer*) e SAML (*Security Assertion Markup Language*). Na abordagem apresentada, quando um provedor de computação em nuvem (chamado de Origem) deseja utilizar recursos de outra nuvem (chamado de Destino) este inicia o processo com trocas de mensagens através do protocolo XMPP (JABBER SOFTWARE, 2004). Esta proposta também implementa os provedores de identidades nas correspondentes nuvens e, portanto, sem os cuidados de segurança necessários. Outros trabalhos fazem o uso do *Shibboleth*, infraestrutura baseada no conceito de federação SAML para organizar as autenticações entre provedores de *clouds* que se associam em federação ((TUSA et al., 2012) e (LEANDRO et al., 2012)) .

Os trabalhos citados acima tratam o gerenciamento de identidade de maneira similar na caracterização de uma federação de *clouds*. Todas porque estão baseadas no SAML, usam o conceito de identidades federadas. E dentre estas propostas, somente a experiência citada em (LEANDRO et al., 2012) é que constrói os provedores de identidades fora das nuvens.

A nossa experiência descrita neste capítulo não usa o conceito de identidades federadas. O nosso modelo é o centralizado em provedores de serviço (que na verdade é a abordagem da ferramenta *OpenID*), ou seja, cada serviço (*SP*) mantém uma lista de provedores que este confia e localiza em qual destes o usuário está registrado usando o identificador *OpenID* do usuário. Uma das vantagens do *OpenID* é que este não requer nenhuma *pilha* de protocolos especiais do lado do cliente. O que não é o caso do *Shibboleth* que requer uma camada de protocolos no lado do cliente, limitando com isto o seu uso em determinados dispositivos móveis.

Em relação ao modelo *IT-IdP* acreditamos ser uma experiência única neste nível. Possíveis intrusões que resultem em comprometimento de *IdPs* (*proxies* no nosso modelo), não resultam na quebra de confidencialidade de dados de usuários e muito menos na interrupção do serviço de autenticação de usuários.

4.11 CONCLUSÃO

Este capítulo descreveu a nossa experiência no desenvolvimento de provedor de identidades *OpenID* tolerante a intrusões. A arquitetura que implementamos faz uso da tecnologia de virtualização e não envolve replicação física do hardware especializado (processadores seguros) que é usado nos *IdP*. As informações de usuários são mantidas em compartimentos separados e qualquer intrusão no modelo se traduz em ações inócuas. A separação dos compartimentos é feita através do isolamento das máquinas virtuais e componentes confiáveis, que se comunicam exclusivamente através de uma memória compartilhada.

O uso da virtualização permite a recuperação da replicação definida já a partir da detecção do comportamento malicioso. Um protótipo foi implementado do modelo desenvolvido e os testes realizados com os mesmo nos dão a certeza da viabilidade das nossas propostas. Nosso trabalho parece singular porque não encontramos experiências correlatas na literatura. As propostas e desenvolvimentos apresentados neste capítulo fizeram parte do Projeto *SecFuNet*.

5 CREDENCIAIS E ATRIBUTOS DE USUÁRIOS EM UMA FEDERAÇÃO DE CLOUDS

Este capítulo mostra a evolução de nossos trabalhos sobre autenticação em federação de *clouds*. O modelo de provedores de identidades, apresentado no capítulo anterior, sofria de restrições de flexibilidade e de escala quando se considerava a evolução de uma federação. O sistema de autenticação era baseado em uma grade de processadores seguros que limitavam mudanças. Neste capítulo, descrevemos um modelo bem mais flexível e também seguro no trato de credenciais e atributos de usuários. Com isto, apresentamos então o novo modelo de Provedor de Identidades tolerante a intrusões mas que faz uso de recursos da federação de *clouds* para armazenar as informações de registro dos usuários. Discutirmos as mudanças no nosso modelo de *IT-IdP* e as técnicas e protocolos usados para armazenar credenciais e atributos de usuários em recursos da federação. Os resultados do modelo implementando são apresentados buscando mostrar a sua viabilidade. Por fim, apresentamos a literatura relacionada a este modelo e as conclusões sobre o modelo.

5.1. MOTIVAÇÃO

Provedores de identidades (autoridades de autenticação) na verdade são pontos centrais na aplicação de políticas de segurança nestes grandes sistemas distribuídos. Defendemos neste texto que, por serem serviços que ficam disponíveis via Internet, estes *IdPs* estão sujeitos a ataques que podem resultar em intrusões, o que seria catastrófico para a segurança das informações e recursos. Inicialmente (BARRETO et al., 2013), apresentado no Capítulo 0, que fez parte do projeto *SecFuNet*, desenvolvemos um provedor de identidades que faz uso extensivo da tecnologia de virtualização para isolar as informações de usuário de ataques maliciosos da rede externa (a *Internet*). Apesar da abordagem citada ter se mostrado factível e eficiente, a manutenção centralizada de informações de usuário (credenciais e atributos) em um serviço de autenticação executado sobre componente seguro, considerando o contexto de grandes sistemas, apresentou pouca flexibilidade e com manutenção complexa.

O serviço de autenticação no *SecFuNet* é construído em cada *IT-IdP* sobre uma grade de processadores seguros, onde credenciais de usuários eram previamente gravadas *off-line* em memória de processadores que depois eram colocados na placa da grid de processadores usados. Se considerarmos uma federação de *clouds*, este sistema de registros envolvendo operações *off-line* se torna inviável.

O objetivo deste capítulo é apresentar uma abordagem que pode ser usada de forma complementar à citada acima, mas que faz uso de provedores de *cloud* para armazenar as informações de usuário (credenciais e atributos), e não mais de componentes seguros. A abordagem introduzida neste capítulo é apresentada como uma proposta de solução para a autenticação em federações de provedores de *cloud*, mas isto não restringe a mesma a estes ambientes. Qualquer *IdP* pode ser concebido usando a mesma abordagem de armazenamento de informações de usuários em recursos de provedores de *cloud* que não formem associações como os das federações citadas.

O uso de *clouds* no armazenamento destas informações de usuário exige um conjunto de mecanismos para fornecer segurança e disponibilidade destas informações, mesmo considerando a falta de controle absoluto sobre estes recursos e serviços delegados. Mas a vantagem desta abordagem está na flexibilidade do acesso às informações permitindo que usuários possam realizar o processo de autenticação a partir de um *IdP* de uma federação de *clouds* e ter acesso a recursos em diferentes partes desta federação. Em outras palavras, queremos dizer que a nossa abordagem de gerenciamento de identidades, pelo uso da especificação *OpenID*, segue o modelo centralizado, mas que pela flexibilidade trazida pela disponibilização das informações nas *clouds*, temos as mesmas características do gerenciamento de identidades federadas, porém sem a necessidade de relações de confiança entre *IdPs* do sistema.

5.2. CARACTERIZAÇÃO DE FEDERAÇÃO DE *CLOUDS* E SERVIÇOS DE AUTENTICAÇÃO

O agrupamento de provedores de *clouds* através de redes de confiança formadas com o objetivo de atender a um vasto número de usuários coloca também grandes desafios na autenticação e autorização destes usuários junto aos provedores nestas federações. A Figura 18 ilustra uma destas associações que visam o compartilhamento de recursos. Nesta figura, explicitamos basicamente o aspecto dos controles de autenticação, centrando sempre estas funções em provedores de identidades (*IdPs*).

Normalmente, os provedores de *cloud* mantêm as funções de autenticação de seus próprios clientes, ou seja, estas *clouds* possuem o que assumimos como *IdPs* internos em suas *clouds*. Mas, atualmente, alguns trabalhos ((CHOW et al., 2012), (NUÑEZ; AGUDO, 2014) e(BERTINO et al., 2009)]) têm assumido as *clouds* como ambientes com segurança limitada para controlar e manipular de forma indiscriminada os dados de seus usuários. Algumas experiências enfatizam o uso de

provedores de identidades fora das nuvens (LEANDRO et al., 2012). Estes *IdPs* externos são mantidos por entidades consideradas confiáveis e liberam as *clouds* de preocupações com a segurança das informações de usuários. Existem também diversas soluções muito difundidas para o gerenciamento de *clouds* como os *frameworks OpenStack* (OPENSTACK, 2016) e *Cloudstack* (CLOUDSTACK, 2016) que permitem a delegação das funções de autenticação para entidades externas (*IdPs* externos) que, por exemplo, executem especificações como a do *OpenID* (OPENID, 2014) ou mesmo outras.

A Figura 18 ilustra um exemplo onde provedores de *clouds* (*Clouds* C_1, C_2 e C_3) apresentam formas diferentes em seus controles de autenticação. O provedor C_1 possui *IdP* interno (IdP_{c1}) que executa suas funções de autenticação. As *clouds* C_2 *e* C_3 ao contrário, delegam suas autenticações de usuários a *IdPs* externos (IdP_{X1}) com os quais mantém relações de confiança. Considerando o arranjo citado, o usuário $User_1$ é mostrado efetuando a sua autenticação junto ao *IdP* de C_1 (IdP_{c1}). Este usuário possui só possui conta na *cloud* C_1 e, diante disto, uma vez autenticado tem acesso a recursos somente do provedor C_1.

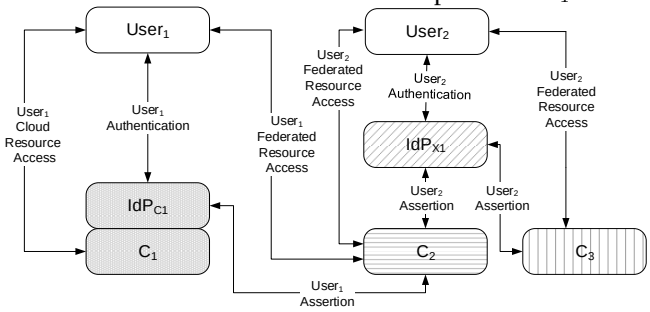


Figura 18 - Autenticação e Autorização em Federação de *Cloud*

Com uma federação composta a partir das *clouds* indicadas na Figura 18, e com as relações de confiança estabelecidas por esta associação, este mesmo usuário ($User_1$) pode também ter acesso a recursos da cloud C_2, uma vez que esta última confia nas autenticações de C_1. Ou seja, os dois provedores de *cloud* (C_1 e C_2) mantêm relações de confiança e, portanto, aceitam as autenticações dos *IdPs* credenciados na federação estabelecida (no caso, IdP_{c1}, IdP_{X1}). Nesta mesma figura, $User_2$ se autentica em IdP_{X1} (um *IdP* externo que C_2 e C_3 confiam) e com isto, este usuário acessa recursos de duas *clouds* da federação (C_2 e C_3). Note que os controles de acesso aos recursos dependem do envio por parte dos

IdPs de asserções de autenticação de usuários aos provedores de *clouds* (*SPs*) que mantêm os recursos desejados.

Na sua forma mais simples, o gerenciamento de identidades no chamado modelo centralizado possui um *IdP* desempenhando o papel de terceira parte confiável e mediando as trocas de clientes e *SPs* no seu domínio de políticas. Em grandes sistemas como *clouds*, outras formas de gerenciamento de identidades são também usadas. Abordagens onde *IdPs* de diversos domínios possuem relações de confiança entre si e autenticações concretizadas em um domínio são aceitas em domínios de outros *IdPs* são conhecidas como "identidades federadas"(JØSANG et al., 2005). Exemplos notórios de infraestruturas que fazem uso de identidades federadas são: o *Shibboleth* (LEWIS, 2008) e o *Liberty Alliance* (LIBERTY, 2003). Protocolos para o gerenciamento de identidades federadas normalmente são baseados nas especificações SAML(OASIS, 2005b), como a infraestrutura *Shibboleth*, formando pilhas complexas de protocolos que devem ser anexadas inclusive nos clientes.

Nossos trabalhos enfatizam o uso de *frameworks* com soluções mais simples para o gerenciamento de identidades, sendo então baseados nas especificações *OpenID* (OPENID, 2014) que seguem o modelo centralizado. Mesmo o modelo centralizado, para poder atender as necessidades de gerenciamento de identidades em grandes sistemas, necessita também estender a escala de suas relações de confiança. Mas diferente do modelo anterior, a extensão das redes de confiança no modelo centralizado não é centrada nos *IdPs* e sim nos provedores de serviço. Ou seja, os *SPs* possuem listas de *IdPs* em quem confiam. Cada provedor de serviços (*SP*) mantém uma lista de *IdP*s em que confia e seus usuários devem ter registros de identidade em pelo menos um destes *IdPs*.

Diante disto, no nosso modelo, para que asserções de autenticação mesmo que remotas sejam aceitas, é necessário que os provedores de *cloud* da federação (*SPs* do nosso sistema) tenham meios de validar as asserções de autenticação que vêm dos *IdPs* da federação. Cada asserção é assinada com a chave privada do *IdP* que autenticou o usuário correspondente e, portanto, cada provedor de *cloud* que faz parte da federação deve gerenciar uma lista de certificados dos *IdPs* da federação (certificados com as respectivas chaves públicas). Neste texto, portanto, enfatizamos o uso de *IdPs* externos às *clouds*.

5.3. ARMAZENAMENTO DE CREDENCIAIS E ATRIBUTOS DE USUÁRIOS

Para a autenticação de usuários, fazemos uso de um *framework* que segue as especificações *OpenID* e que, como já dito neste texto,

implementa o modelo centralizado de gerenciamento de identidades. Estes protocolos definidos na especificação *OpenID*, portanto, não enfatizam relações de confiança entre *IdPs* como nas federações SAML. Com o uso do *OpenID Connect*, os provedores de serviço devem gerenciar suas listas de *IdPs* com quem mantêm relações de confiança e dos quais aceitam asserções de autenticação.

Neste capítulo, a nossa proposta está baseada na existência de diversos provedores de identidades credenciados na federação. Porém, diferentemente de outras abordagens, em nosso modelo de gerenciamento de identidades os atributos de usuários não são armazenados diretamente nos provedores de identidades. A ideia é dispor estas informações, de forma segura, em uma base de dados construída sobre recursos distribuídos pela federação. Porém, manter as informações armazenadas em recursos distribuídos na federação, logicamente, envolve vários problemas. Com as informações de usuários em uma base de dados distribuída em *clouds* que caracterizam ambientes cujos recursos computacionais são administrados e acessados fisicamente por terceiros, implica na necessidade de garantir a segurança e a disponibilidade destas informações. As soluções para este problema quase sempre envolvem métodos de criptografia e replicação das informações dos usuários. A criptografia depende também da segurança das chaves criptográficas usadas.

O armazenamento seguro em *clouds* tem como solução usual a encriptação dos dados, o que garante a confidencialidade dos mesmos. As chaves criptográficas usadas nestas encriptações são protegidas normalmente com o uso de técnicas de compartilhamento de segredos (SCHOENMAKERS, 1999; SHAMIR, 1979). As informações criptografadas podem ser replicadas e armazenadas em recursos de várias *clouds* de modo a garantir a disponibilidade das mesmas em ambientes onde podem ocorrer ações maliciosas ou mesmo falhas. Porém, existem soluções mais baratas que fazem uso de técnicas de dispersão de informação (*Information Dispersal Algorithms*: *IDA* (RABIN, 1989)), baseadas em códigos de correção de erro (*erasure codes*), que diminuem os custos de armazenamento ao de uma simples cópia dos dados. Estas transformações *IDA* particionam os dados de uma informação em um conjunto de n partes (ou blocos) de tal maneira que t partes (com $t < n$) são suficientes para que se recupere a informação original. Com $t - 1$ ou menos partes os dados, como um todo, não podem ser recuperados. No entanto, o uso simples destas técnicas de recuperação de erros não garante a confidencialidade das informações; é necessário a encriptação das partes das mesmas.

Em um esquema de criptografia de limiar, chamado de compartilhamento de segredo, a partir de transformação, um segredo *S* dá origem a *n shadows* (partes). Com isto, cada uma destas partes do segredo (ao contrário das partes geradas por uma *IDA*), não permite qualquer inferência sobre o segredo *S*. Esta informação só pode ser reconstruída a partir da obtenção de um conjunto mínimo de *t* partes ($0 < t \leq n$). Desta forma, em posse de qualquer conjunto de *k* partes do segredo, com $k < t$, não se consegue recuperar informação alguma de *S*. Os primeiros esquemas de compartilhamento de segredo propostos na literatura (SHAMIR, 1979) definiam compartilhamentos de segredo baseados em interpolação polinomial e geometria plana (intersecção de retas). Estes esquemas requerem sempre um entregador honesto na inicialização da computação para gerar as partes do segredo e distribuí-los aos *n* participantes envolvidos no compartilhamento.

Para lidar com cenários onde as partes estão sob a ação de entidades maliciosas que agem ativamente contra os protocolos de restauração dos segredos, foram criados os mecanismos de compartilhamento de segredo verificável (*VSS*), em que cada fragmento de um segredo ou computação fornecido a um combinador pode ter sua validade verificada. Estes esquemas verificáveis são fundamentais para que o resultado de uma recuperação de segredo não passe por valores errôneos devido a partes incorretas fornecidas por participantes maliciosos, acelerando deste modo a obtenção do resultado correto da computação. Um esquema interessante é o compartilhamento de segredo verificável publicamente (*publicly verifiable Secret Sharing scheme - PVSS*) (SCHOENMAKERS, 1999), onde não apenas os *n* portadores de partes de segredos podem verificar a validade dos mesmos, mas também qualquer parte que interaja no protocolo (por isso o "verificável publicamente").

Vários sistemas que por armazenarem arquivos em *clouds* baseados em transformações *IDA* e seus códigos de correção de erro (BESSANI et al., 2011; KUMAR et al., 2012; SUJANA et al., 2013), precisam que seus arquivos de informações sejam criptografados, para garantir a confidencialidade dos mesmos. As chaves usadas na encriptação das informações também precisam ser protegidas, e neste caso são aplicados esquemas de compartilhamento de segredo. Em um provedor de identidades, as informações são de poucos *bytes* (credenciais e atributos de usuários) e não se torna proibitivo a aplicação de esquemas de compartilhamento de segredo diretamente nestas informações. Portanto, não há necessidade de adoção de técnicas de *erasure codes* e da

criptografia adicional para garantir a confidencialidade destas informações.

Na nossa abordagem, portanto, aplicamos somente um esquema de compartilhamento de segredo sobre as informações de usuário que decompõe as mesmas em *n* partes. Estas partes são então distribuídas entre *n* diferentes entidades (uma parte em cada uma das *n* entre as *m clouds* do sistema). As informações são recuperadas a partir de no mínimo *t* partes que estão armazenadas em *n* das *m clouds* do sistema, onde $0 < t < n < m$. Nas experimentações da nossa abordagem, usamos dois esquemas de compartilhamento de segredo: um não verificável e mais simples (SHAMIR, 1979) e o *PVSS* (SCHOENMAKERS, 1999).

Portanto, na abordagem proposta, os atributos de usuários não são armazenados diretamente nos provedores de identidades, mas sim em uma base distribuída e segura (por conta do compartilhamento de segredo). Os *IdPs* não se preocupam com a manutenção destas informações de usuários, mas sim em executar os protocolos e especificações para a criação de asserções ou *tokens* de autenticação.

5.3.1. Premissas Assumidas

O fato de colocarmos os nossos dados pessoais em recursos gerenciados por terceiros envolve uma preocupação com a confidencialidade e a integridade destes dados. Alguns autores assumem que provedores de *cloud* mantêm um comportamento honesto, mas curioso (*honest-but-curious*) (GOLDREICH; MICALI; WIGDERSON, 1987). Este tipo de comportamento limita as ações de entidades de uma *cloud* a apenas leitura de informações em dados armazenados nos recursos da *cloud*. Porque são honestas, estas entidades não repassam estes dados e não fazem conluios com entidades de outras *clouds* da federação em tentativas de reconstruir informações dos segredos.

Em nossa abordagem, assumimos premissas menos restritivas. Cada provedor de *cloud* pode ser confiável, mas entidades maliciosas podem agir internamente devido à grande disponibilidade dos recursos de uma *cloud* a acessos externos ou mesmo internos. Estas entidades podem formar conluios para quebrar segredos. Os segredos dos usuários são mantidos, em nossa abordagem, quando o número de *clouds* sofrendo intrusões e falhas no sistema não ultrapassa o limiar *f* que definimos como sendo de valor $n - t$ ($f = n - t$). Nesta situação nossos protocolos funcionam corretamente e os segredos das informações não são quebrados e podem ser seguramente transformados.

Mantendo os atributos de usuários armazenados de forma distribuída torna possível que qualquer provedor de identidades da federação que execute os protocolos de disseminação, tenha possibilidade

de recuperar, combinar as informações de seus usuários para verificar as credenciais de seus usuários e gerar as asserções de autenticação correspondentes.

5.3.2. Organização dos IT-IdPs para Compartilhamento de Segredos

A proposta apresentada nesta seção mantém características similares a abordagem do modelo descrito no Capítulo 0, mas apresenta modificações que são efetuadas no sentido de integrar os algoritmos de compartilhamento de segredo à arquitetura do serviço de autenticação tolerante a intrusão. Uma visão geral do modelo é ilustrada na Figura 19, onde são explicitadas as funcionalidades em provedores de identidades, provedores de *clouds* e ferramentas de administração necessárias para o armazenamento baseado em segredo de credenciais e atributos de usuários. Cada provedor de identidades (*IdP*) mantém as mesmas pilhas e executa as mesmas funcionalidades.

Assim como descrito no modelo apresentado no Capítulo 4, os protocolos *OpenID* são executados por *IdP* proxies isolados em suas máquinas virtuais (*VMs*). As interações com entidades externas do IT-IdP (*browsers* e *SPs*), portanto, não se alteram e continuam sendo executadas segundo os protocolos *OpenID*. As operações e disposição do Repositórios de Descritores (*RD*) são também mantidas idênticas ao do modelo ado capitulo anterior. As relações entre as entidades da zona confiável continuam as mesmas (Memória Compartilhada e o *Agreement Service*).

As diferenças estão no Serviço de Autenticação (*AuS*) que é a entidade responsável pelas verificações durante os processos de *login* dos usuários. Este serviço não mais é executado sobre *hardware* especializado (não está mais baseado em *grid* de processadores seguros, como no capítulo anterior). Agora, o AuS se apresenta isolado, construído dentro da *trusted zone* do modelo, junto do *Agreement Service*.

Quando disseminadas entre recursos da federação, é no espaço do Serviço de Autenticação que se faz a recuperação das informações de usuário (credenciais e atributos). Estas informações, obtidas através de algoritmos de compartilhamento de segredos, são usadas pelo *AuS* na validação de credenciais e para atender demandas de atributos. Por restrições de desempenho devido a questões de rede e ao próprio custo dos algoritmos envolvidos, o Servidor de Autenticação (*AuS*) tem disponível internamente uma *cache* para manter estas informações recuperadas. Esta *cache* é justificada pela manutenção dos atributos e credenciais de usuários mais recentes e frequentes, evitando sempre que

possível à execução dos protocolos de combinação e a recuperação das informações de usuários.

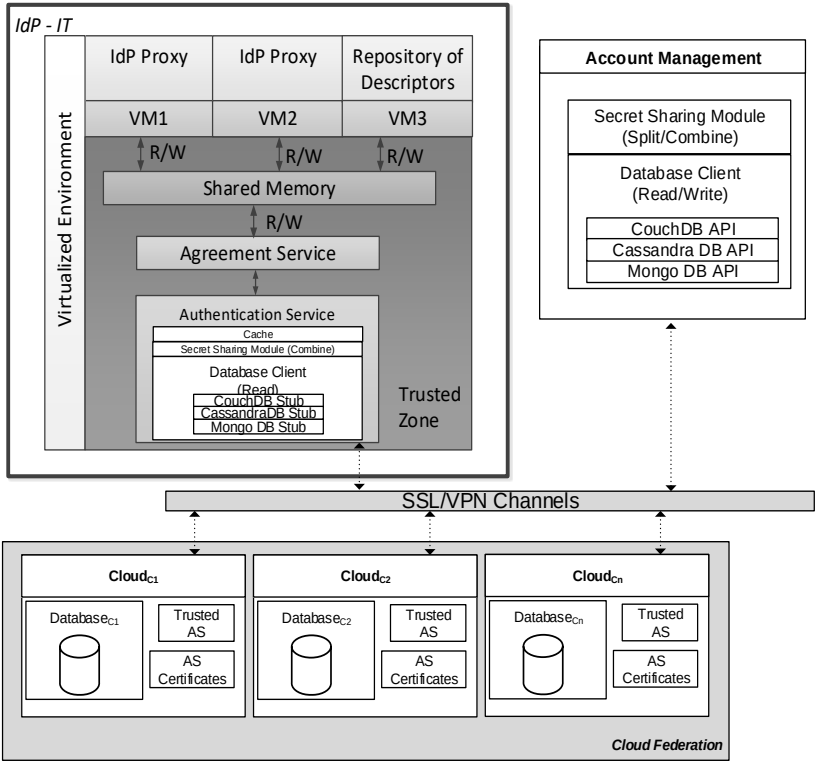


Figura 19 – Organização do Sistema de Autenticação

Nos domínios do *AuS*, estão também as funcionalidades da camada *Secret Sharing* que combinam as partes dos segredos que estavam espalhados pelas *clouds*, deixando as informações necessárias de usuários disponíveis na *cache* para as verificações de *login* (Figura 19).

Para que esta camada de compartilhamento de segredos (*Secret Sharing*) possa obter as partes de um segredo referente a um usuário na federação de *clouds*, é necessário o uso de *client stubs* para a interação com as diversas ferramentas que compõem a base de dados distribuída de armazenamento destas partes. Na Figura 19, estas *stubs* são apresentadas como parte da camada *Database Client*. Cada *stub* é correspondente a uma das diferentes bases de dados, uma vez que a diversidade de programação (AVIZIENIS et al., 2004) é uma imposição do ambiente heterogêneo e sujeito a intrusões como uma federação de *clouds*. A

camada *Database Client* possui, portanto, uma interface genérica que oferece a operação de *read* para a obtenção das diferentes partes do segredo, deixando transparente para a camada superior (*Secret Sharing*) as especificidades de cada ferramenta de banco de dados usada na federação.

Na execução do processo de autenticação, o *AuS* se utiliza desta pilha de camadas (*cache*, *Secret Sharing* e *Database Client*) de forma análoga ao acesso a uma base de dados, requisitando (*querying*) então as informações do usuário distribuídas nas *clouds*, para a verificação de credenciais apresentadas ao mesmo durante uma requisição de *login*.

Como tratamos com informações confidenciais, é imprescindível a garantia de isolamento do *AuS* em relação à rede externa (Internet). Também nas trocas entre provedores de identidades e *clouds* da federação para o armazenamento de partes de segredos, é necessário a utilização de canais confiáveis. Estes canais seguros de comunicação são ilustrados na Figura 19. Além de fazer uso de mecanismos criptográficos estes canais são implementados através do uso de redes privadas virtuais e são mantidos única e exclusivamente para a comunicação entre as bases de dados das *clouds* da federação e os *AuS* presentes nos diversos provedores de identidades de nossa abordagem.

Além da implementação das especificações para autenticação de usuários através dos protocolos *OpenID*, nossa arquitetura estende a infraestrutura com interface e pilha de protocolos necessárias para a criação de registros dos usuários. A camada *Account Management* representada na Figura 19 permite então que registros com credenciais e atributos de usuários sejam criados em base de dados distribuída na federação. As operações da interface desta camada permitem a introdução ou alteração de informações de usuários.

Esta camada *Account Management* faz, portanto, um uso mais amplo da camada *Secret Sharing*. Além da combinação de partes de um segredo, algoritmos de geração de partes de um segredo são fornecidos a esta camada. Portanto, os atributos e credenciais passados durante a criação de uma conta são transformados em partes de segredos através da camada *Secret Sharing* da camada *Account Management*. Por sua vez, a camada *Database Client* fornece os acessos (*read*/*write*) aos serviços de ferramentas de bancos de dados das diversas *clouds* da federação, para o armazenamento e a recuperação das informações de usuários durante a criação e atualização de suas contas.

A Figura 19 mostra ainda os provedores de *cloud* da federação. Em relação ao sistema de autenticação, estes provedores de *cloud* tornam disponíveis as ferramentas para os serviços de base de dados distribuída

que armazena as partes de segredos referentes às credenciais e atributos de usuários.

5.4. PROTOCOLOS PARA O ARMAZENAMENTO DE INFORMAÇÕES DE USUÁRIO

5.4.1. Registro de Usuários

Como introduzido na seção anterior, os registros de usuários devem ser efetuados através da interface do serviço Account Management de um IT-*IdP* de *cloud* presente na federação. Na criação de contas, assumimos duas possibilidades: na primeira, a interface Account Management é acionada por agente credenciado pela federação de *clouds* e é necessária a presença física do usuário; na alternativa, o próprio usuário faz o seu registro através da Internet. Em ambos os casos, as credenciais e atributos do usuário são introduzidos através de um formulário *web*.

Na criação de conta com a exigência de presença física do usuário, o cadastro é efetuado segundo os conceitos de segurança definidos em (BURR; DODSON; POLK, 2006), onde a validade da identidade assegurada é de nível *LoA 4* (*Level of Assurance 4*). Para este nível, o usuário deve comparecer junto ao agente credenciado, com os documentos necessários para garantir a veracidade de suas informações em seu registro. O agente então, com base nestes documentos, introduz os dados do usuário no sistema.

Na alternativa, para a criação de contas por acesso remoto à interface da camada *Account Management*, o usuário deve fornecer um certificado de chave pública assinado por autoridade certificadora oficial (*CA* de uma *PKI* oficial), ou ainda, por uma *CA* reconhecida pelos provedores de *cloud*. A garantia da validade da identidade apresentada está na verificação da vinculação desta identidade com a chave pública do certificado apresentado. Nesta última alternativa, o usuário ainda fornece ou altera seus atributos e credenciais, devidamente assinados por sua chave privada. Esta possibilidade de criação de contas, sem a presença física de usuários, está de acordo com o nível *LoA 3* do guia sobre autenticação eletrônica (BURR; DODSON; POLK, 2006).

As trocas do protocolo de criação de conta de usuário (segundo o nível *LoA 3*) são ilustradas na Figura 20. Em um primeiro passo, o usuário preenche o formulário de criação de conta através de um *IT-IdP* da federação, introduzindo suas informações para o seu registro no sistema (passo 1, na Figura 20). Os dados passam por uma validação local (verificação da correção do certificado do usuário e também das assinaturas sobre os atributos passados pelo usuário).

Uma vez validadas, as informações de usuário (credenciais e atributos), estas são passadas à camada *Secret Sharing* para que sejam geradas as partes do segredo e disseminadas entre as *clouds*, na base de dados distribuída que vai alimentar os *IT-IdP*s do sistema. Este processo envolve algumas definições como: a escolha no esquema de compartilhamento de segredo, definindo o número de partes do segredo e o limiar (número mínimo de partes) para a recuperação dos segredos, dentre outros (passo 2).

Após a geração das partes do segredo a partir das informações do usuário (passo 3), a distribuição destas se dá através das *stubs* apropriadas da camada *Database Client* (passos 4 na Figura 20). O número *n* de *clouds* usadas para o armazenamento destas informações é menor que o número total (*m*) de *clouds* na federação. Estas distribuições envolvem a definição de *keys* (identificações únicas dos registros de usuários). Na nossa abordagem, adotamos como estas chaves os próprios identificadores *OpenID* de cada usuário. Estas chaves são usadas pelos *IT-IdP*s, posteriormente, através de seus Serviços de Autenticação (*AuS*), na recuperação das diversas partes do segredo. A escolha das *clouds* que deverão ser usadas para o armazenamento (*n* entre as *m* disponíveis) é feita aleatoriamente, evitando, porém, que partes de um mesmo segredo sejam armazenadas na mesma *cloud*.

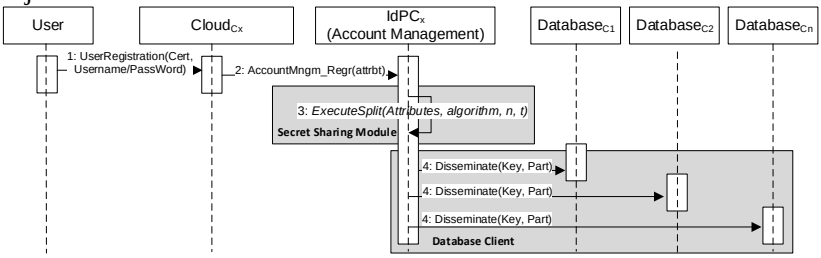


Figura 20 - Registro de Usuários

5.4.2. Autenticação de usuários

O processo de autenticação neste trabalho, por usar os protocolos *OpenID*, não difere dos *logins* usuais de *IdP*s. A grande distinção está na forma com que estes provedores de identidades têm acesso aos atributos de usuário. Uma descrição do protocolo de autenticação de usuário é apresentada na Figura 21.

Este procedimento é iniciado quando o usuário tenta acessar um provedor de *cloud* para requisitar recursos (passo 1, Figura 21) que, por sua vez, verifica se este usuário mantém alguma sessão de computação com autenticação já válida (ou seja, se possui recursos ou serviços já

alocados). Caso não tenha, a *cloud* estabelece comunicação com um dos *OpenID* proxies do *IT-IdP* indicado pelo identificador *OpenID* do usuário (passo 2). Se o *Agreement Service* (*AgS*) deste *IdP* possui uma asserção de autenticação válida do usuário, no caso de o mesmo já estar autenticado e acessando recursos em outra *cloud* (passo 3), a asserção correspondente é então enviada ao provedor de *cloud* requisitante (passo 4). Caso esta asserção de autenticação também não exista no *IT-IdP* contatado, então o usuário é direcionado para este provedor de identidades (o *IT-IdP* indicado pelo seu identificador *OpenID*) para que sua autenticação se concretize.

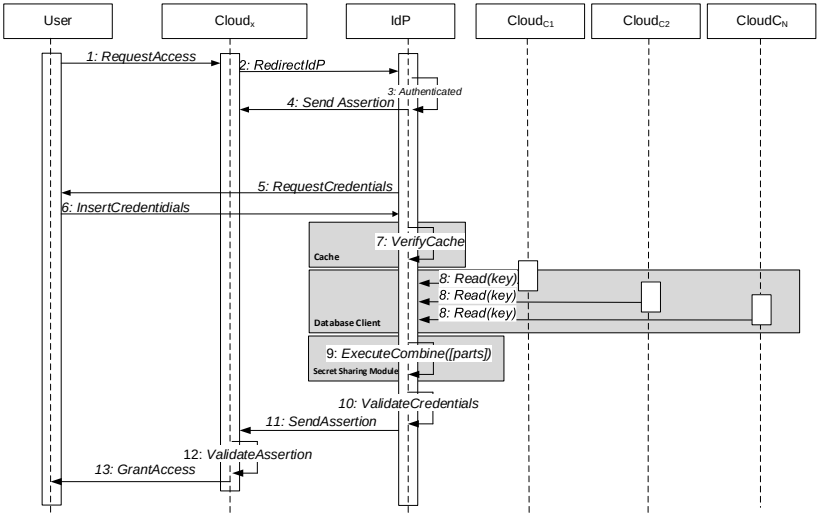


Figura 21 – Autenticação de usuários

O processo de autenticação é iniciado com o *IT-IdP*, através de um de seus *proxies*, solicitando ao usuário suas credenciais de autenticação (passo 5). Após a inserção destas credenciais (passo 6), o Serviço de Autenticação (*AuS*) do *IT-IdP* faz consulta para verificar se as informações de *login* do usuário estão em sua *cache* (passo 7). Se estas informações estiverem presentes, o *AuS* do *IT-IdP* confronta estas informações com as credenciais fornecidas, validando o usuário e permitindo a geração de uma asserção de autenticação (passo 10). Caso não estejam disponíveis na *cache*, o *AuS* deve buscar as informações do usuário na base de dados distribuída entre *clouds* da federação (passo 8). Esta busca na federação faz uso do identificador *OpenID* do usuário como *key* na busca de seu registro pessoal na base de dados.

A busca de informações do usuário na federação faz uso da camada *Database Client* para a recuperação das partes de segredo que foram distribuídas entre *clouds* da federação. Com *t* partes do mesmo segredo, a camada *Secret Sharing* faz a combinação e recupera as credenciais e atributos do usuário correspondente, tornado disponíveis as mesmas na *cache* do *IT-IdP* (passo 9). De posse destas informações, o *AuS* faz as validações necessárias sobre as credenciais informadas pelo usuário (Passo 10) e, uma cópia da asserção gerada no processo de autenticação é enviada ao provedor de *cloud* indicando o sucesso ou não das verificações (passo 11).

A Figura 21 mostra também a recepção da asserção de autenticação pelo provedor de *cloud* que efetua a validação da mesma com as verificações dos *nonces*, data de validade da asserção e assinaturas (passo 12). É importante ressaltar que o *IT-IdP* em questão deve estar na lista de *IdP*s confiáveis do provedor de *cloud* (possui o certificado de chave pública deste *IT-IdP*). Com a asserção válida, o provedor de *cloud* (o *SP* das interações do caso) inicia o tratamento da requisição do usuário diante de sua política de autorização (Passo 13).

O fato do usuário se registrar junto a um *IT-IdP* (uso da interface Account Management) deixa o usuário ligado a este provedor de identidades pelo seu identificador *OpenID*. Mas o usuário mesmo assim no nosso modelo tem a sua identidade assumindo propriedades próximas da abordagem de identidades federadas: o login feito em um *IdP* de cloud vale em todas as *clouds* da federação.

5.5. PROTÓTIPOS E RESULTADOS

Com o objetivo de verificar a viabilidade de nossas propostas, um protótipo foi desenvolvido onde um provedor de identidades seguindo o modelo do *IT-IdP* descrito nas seções 4 e 5, e também baseado nas especificações *OpenID*. O protótipo permite a autenticação de usuários, o armazenamento das informações de usuários em *clouds* da federação conforme preconizado nas seções citadas. As asserções de autenticações geradas nos processos de *login* podem ser usadas por provedores de *cloud* de toda a federação. Além da asserção de autenticação, a *cloud* da federação pode requisitar atributos do usuário (nome, endereço, etc.) nas verificações de controle de autorização do usuário que se autenticou no sistema.

A Figura 22 descreve uma visão das ferramentas utilizadas na implementação do protótipo do sistema de autenticação. No nível mais alto, exposto à *Internet*, estão os *IdP proxies* que seguem as especificações do *OpenID* em seus protocolos de interface com usuários e provedores de *cloud*. Os serviços disponíveis a partir desses *proxies*

servem somente para a autenticação de usuários, pois o registro de usuário, como citado no texto, é executado por extensão (a interface *Account Management*), em pilha de protocolos própria a partir dos provedores de identidades.

A replicação dos *IdP* proxies foi implantada através do software de virtualização XEN, para a execução das máquinas virtuais. Além disso, o uso do *hypervisor* XEN oferece a facilidade de implementação da memória compartilhada para a comunicação entre as *VMs proxies* e o *Agreement Service* (*AgS*). A memória compartilhada e o *AgS* foram implementados no nível do *hypervisor*, o que permitiu o isolamento destas abstrações, de forma que ambas possam ser consideradas componentes confiáveis, que é premissa básica da proposta. Nas *VMs* de proxies foram utilizadas diferentes versões do sistema operacional *Linux*, com o objetivo de manter certa independência de vulnerabilidades já que estas máquinas virtuais são os únicos componentes visíveis externamente (via rede).

O Serviço de Autenticação é também mantido isolado pelo *hypervisor* XEN. Este componente é composto por diferentes camadas (*Cache*, *Secret Sharing* e *Database Client*), com as funções da execução do protocolo de compartilhamento de segredo e da disseminação das informações nas diferentes *clouds* da federação. Na camada *Secret Sharing* que provê as funcionalidades de compartilhamento de segredos, foi usada na sua implementação o esquema *PVSS*, com as devidas bibliotecas (BESSANI, 2006). A nossa implementação desta camada permite a definição dos parâmetros (*n*, *t*) do algoritmo de compartilhamento de segredo. A implementação do *Database Client* provê funcionalidades de escrita e leitura das partes, provendo interfaces (*stubs*) de diversas bases de dados (*MongoDB* (MONGODB, 2016), *CassandraDB* (APACHE, 2016a) e *CouchDB* (APACHE, 2016b), disponíveis através de *clouds* como serviços.

Para a implementação da *Cache* do modelo utilizamos a ferramenta *Infinispan* (REDHAT, 2016). A política implementada para *cache* é de permanência de informações mais frequentemente acessadas. Quando novas informações de usuário são requisitadas, a *cache* é o local em que estas ficam disponíveis na sua forma natural. Se a memória *cache* está totalmente preenchida, os novos dados devem ocupar o local de dados mais antigos e de acessos menos frequentes. Definimos então esta *cache* para ser o espaço de memória para as buscas iniciais do *IdP*. A validação de credenciais em qualquer autenticação e a necessidade de atributos no processo de autorização determinam sempre buscas iniciais nesta *cache*.

A adoção da ferramenta *Infinispan* nos permite definir dois tipos de tempos de permanência dos dados em seus armazenamentos na *cache*: com o parâmetro *lifeSpan* pode ser definido o tempo máximo de vida do dado em *cache*; e com *maxIdle* é definido o tempo de vida em *cache* de dados baseado na frequência de seus acessos.

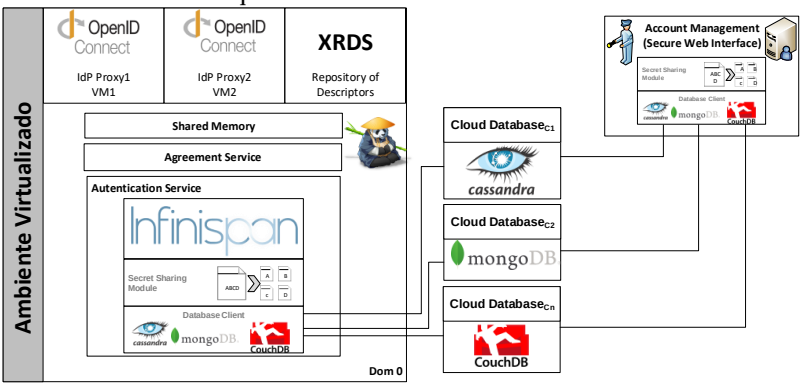


Figura 22 – Tecnologias utilizadas

Os atributos e credenciais dos usuários registrados no sistema são armazenados então em diferentes provedores de *cloud* que oferecem serviços de base de dados. A Camada *Account Management* neste protótipo foi implementada também em uma máquina virtual e, através de suas versões das camadas *Secret Sharing* e *Database Client*, permite a criação e atualizações destes registros de usuários. A interface do sistema foi implementada como uma aplicação *web* onde, através de formulários, os usuários inserem seus atributos e credenciais. Por se tratar de um serviço acessível através da internet e não ter como requisito a presença física do usuário. Os registros de contas via esta interface são classificados como nível *LoA 3*.

5.5.1. Testes e Resultados

Os testes realizados com o protótipo do sistema de autenticação proposto envolveram as operações de autenticação de usuários e a geração das asserções de autenticação correspondentes. Para a realização destes testes, uma quantidade de dados foi gerada, contendo as credenciais de usuários (usuário, senha e certificados) e atributos (nome, sobrenome, endereço, data de nascimento, profissão, etc.) de 1000 usuários. Estes dados gerados passaram pelos algoritmos de compartilhamento de segredo e, posteriormente, foram disseminados no sentido de preencher as bases de dados das *clouds*.

As diferentes *clouds* receberam partes de segredos, levando em conta diversas combinações de parâmetros necessários para o algoritmo

usado no compartilhamento de segredo. A Tabela 8 apresenta os casos de testes que aplicamos na base de dados distribuída gerada a partir das informações de usuários (como citado acima). Os parâmetros *m*, *n* e *t* que aparecem nesta tabela, são usados nos algoritmos de compartilhamento de segredo durante os diferentes testes realizados, sendo *m* o total de *clouds* disponíveis, *n* o total de partes da distribuição de segredos e *t* o número mínimo de partes para a recuperação de um segredo (as informações de um usuário). Os valores escolhidos foram determinados pelas condições de acesso a *clouds* que tivemos e por se mostrarem significativos em relação às condições desejadas de disponibilidade e de segurança. Para nossos testes utilizamos máquinas virtuais da Microsoft Azure, Google Cloud e a infraestrutura de cloud presente em nossa rede interna.

Tabela 8 - Parâmetros de m, n e t dos testes realizados

M	*N*	*T*	**Descrição**
12	*6*	*3*	Registro particionado em 6 partes com no mínimo 3 para recuperação.
12	*9*	*6*	Registro particionado em 9 partes com no mínimo 6 para recuperação.
12	*12*	*10*	Registro particionado em 12 partes com no mínimo 10 para recuperação.

São três os intervalos de tempo considerados relevantes em nosso sistema. O primeiro é o *Register Time*, que é o tempo gasto na introdução de um novo usuário na base de dados de nosso *IdP*. O outro tempo importante de nosso sistema é o *Querying Time* que corresponde ao intervalo de tempo entre a demanda de dados de usuários feita pelo Serviço de Autenticação (querying) e a disponibilidade dos mesmos em *cache* no *AuS*. O terceiro intervalo de tempo considerado é o *Client Response* Time que é o período de tempo entre a inserção de credenciais pelo usuário e o envio de asserção de autenticação correspondente ao *SP*. Estes três intervalos de tempo estão ilustrados na Figura 23. O *Register Time* envolve a subdivisão *tpss* que corresponde ao tempo gasto no particionamento dos atributos de usuário, através dos algoritmos da camada *Secret Sharing*; e a subdivisão t_w que é o intervalo de tempo despendido na escrita das *n* partes do segredo em *clouds* da federação. Por sua vez, o *Querying Time* em sua pior situação, apresenta as seguintes subdivisões: o t_c que é o tempo de procura em *cache*; o t_{get} que corresponde ao tempo gasto na busca das partes nas *clouds*; e, por fim, o t_{pss} que é o tempo necessário para a reconstrução dos dados particionados utilizando os algoritmos de *Secret Sharing*. Na melhor situação o *Querying Time* é igual ao t_c (tempo de procura na *cache*). O *Client Response Time* apresenta possui três subdivisões: o tp_{OpenID} é o tempo referente ao envio das credencias pelo usuário via protocolo *OpenID* ao Serviço de Autenticação; o *Querying Time* que recupera segredos para a

verificação das credenciais dos usuários e o t_A que corresponde ao tempo para envio ao *SP* da asserção de autenticação.

Automatizamos os testes através da utilização da ferramenta JMeter (APACHE, 2016c) e retiramos nossos resultados dos sumários gerados por essa ferramenta. Na sequência, apresentamos estes resultados dos testes.

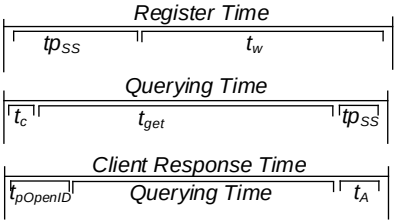


Figura 23 –Tempos Verificados nos Testes

5.5.2. Tempo de Criação de Contas de Usuários (Register Time)

Neste teste foi medido o tempo necessário para a geração das partes de segredo pelos algoritmos de compartilhamento (tp_{SS}) e também no armazenamento dos dados resultantes nas diversas *clouds* da federação (t_w), representando o registro de usuários através da interface *Account Management*. A Figura 24 apresenta os tempos médios de registro obtidos em nossos testes onde podemos verificar a variação entre 350 e 550 *ms* para estes tempos considerando as variações dos parâmetros do compartilhamento de segredo como $6 \leq n \leq 12$ e $3 \leq t \leq 10$. Na Figura 24, o t_w permaneceu relativamente constante com valores em torno de 300 *ms*. O tempo a geração das partes na camada *Secret Sharing* é que apareceu como altamente dependente dos parâmetros usados, variando entre 300 e 600 *ms*.

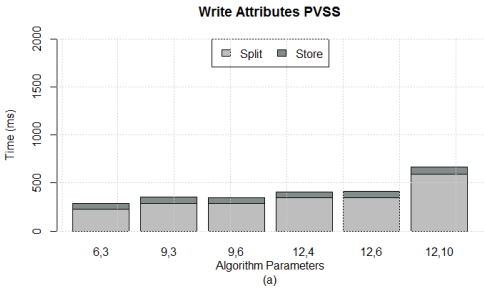


Figura 24 – Tempos de Registro

5.5.3. Testes sem o uso de cache e sem intrusões

Os testes neste item são simplificados, não considerando o uso de *cache* e também não ocorrem informações corrompidas por possíveis intrusões no sistema.

5.5.3.1. Tempos de Consulta (Querying Time)

Na obtenção de valores de *Querying Time*, consideramos primeiro uma pequena carga de utilização (10 usuários simultâneos) e, posteriormente, uma carga de utilização bem mais significativa (100 usuários simultâneos).

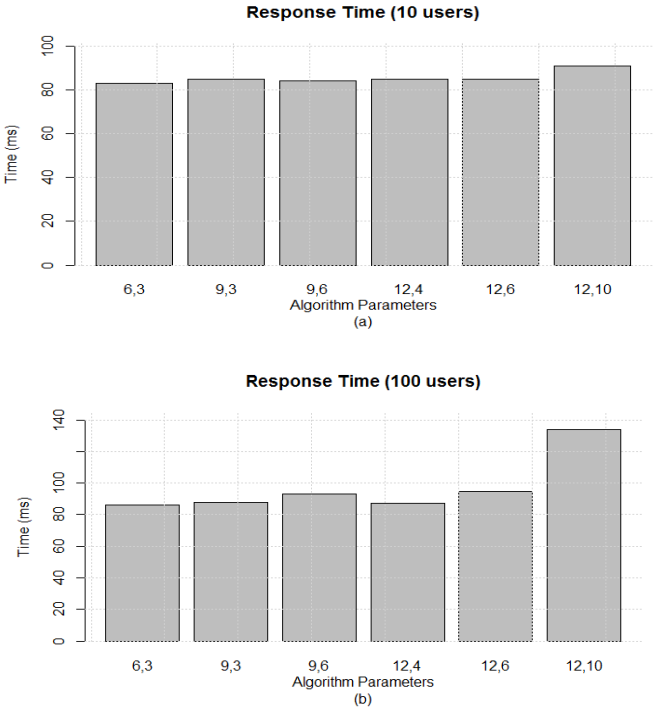


Figura 25 – Tempo de Querying

A Figura 25 apresenta estes tempos envolvendo também diferentes valores dos parâmetros (n, t) do protocolo de compartilhamento de segredo. É possível perceber que o *Querying Time* apresenta pequenas variações nas diferentes configurações quando consideramos dez (10) usuários simultâneos (Figura 25a). O mesmo não acontece nos testes de cem (100) usuários simultâneos (Figura 25b). As leituras nas diferentes bases de dados das *clouds* acontecem de forma paralela, de modo que o

Querying Time será sempre determinado a partir da *cloud* que responde mais lentamente.

Diante dos valores de *Querying Time*, a parcela correspondente ao tempo gasto na camada *Secret Sharing* na recuperação do segredo (tp_{SS}) não é muito significativa. Os gráficos da Figura 26 mostram que a partir de 100 usuários sendo atendidos simultaneamente levam estes tempos no geral a se aproximarem de 4,6 *ms*.

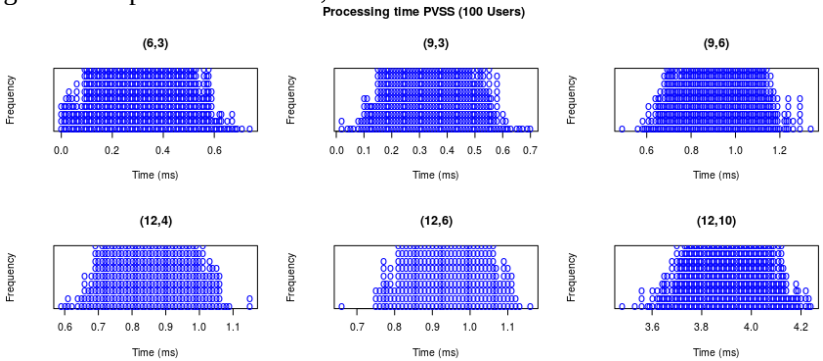


Figura 26 – Tempo Processamento na Camada *Secret Sharing*

5.5.3.2. Tempo de Resposta ao Cliente (Client Response Time)

Neste teste verificamos o tempo de resposta do processo de autenticação no lado do cliente também não considerando dados em *cache* e também sem dados corrompidos por intrusões. Para estes testes, consideramos o sistema com uma carga constante de 100 usuários se autenticando repetidamente, e com um dos usuários medimos o seu *Client Response* Time. Como o processo de autenticação requer a inserção de credenciais (usuário/senha), a automatização do envio destas informações é feita via o método POST através da ferramenta JMeter. A Figura 27 apresenta histograma com valores de *Client Response* Time para as diferentes configurações do algoritmo de compartilhamento de segredo, conforme indica a Tabela 8.

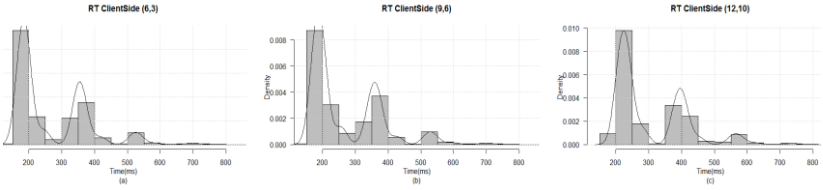


Figura 27 – *Tempo de resposta no cliente*

De forma geral, nas configurações (6,3) e (9,6), estes tempos de resposta ficam agrupados em sua maioria em torno do valor de 150 *ms*. Na configuração (12,10) do compartilhamento de segredo, os valores *Client Response* Time ficam em sua maioria próximos de 230 *ms*.

5.5.4. Tempo de Resposta ao Cliente (*Client Response Time)*

Nestes testes para a obtenção de valores de *Client Response Time* é considerado o uso de *Cache* no Serviço de Autenticação. Também consideramos uma porcentagem dos atributos dos usuários como presentes em *cache* e, portanto, sem a necessidade da busca dos mesmos nas diferentes *clouds* da federação (o *Querying Time* é igual ao tempo de *cache*). É possível perceber no histograma da Figura 28 que o *Client Response* Time com *cache* em seus valores, diminui em média 130 *ms* se comparados com os mesmos casos de testes quando a *cache* não foi utilizada (seção anterior). De forma geral. Os tempos de busca (*Querying Time*) dos atributos na *cache* leva aproximados de 1 *ms* que são de duzentas a trezentas vezes menores daqueles obtidos quando as buscas envolvem a recuperação nas *clouds* da federação.

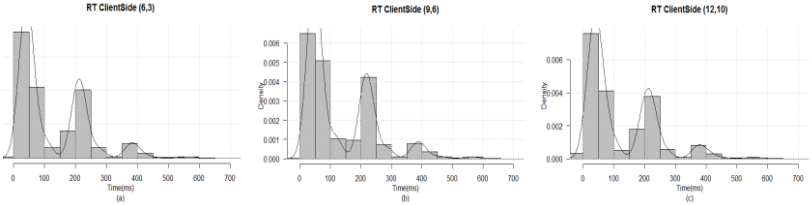


Figura 28 – *Client Response* Time (*Cache*)

5.5.5. Testes com Intrusões e sem Dados em Cache

a) *Client Response* Time

Nas seções anteriores foram discutidos os resultados da execução de nossa proposta em um ambiente sem intrusões onde os *IdPs Proxies* apresentam seu comportamento normal, executando o protocolo e entregando ao SP as asserções de autenticações dos usuários do sistema. Nesta seção buscamos avaliar as implementações nos casos onde o sistema é comprometido através de intrusões, fazendo com que *IdP* proxies apresentem comportamentos arbitrários em suas execuções das especificações do *OpenID*. Consideramos nestes testes, como somente um dos *IdP* proxies pode ser comprometido. Se as duas réplicas proxies fossem assumidas como comprometidas envolveria então a necessidade de uma nova descoberta através do SP (a busca de um novo arquivo *XRDS* com a nova configuração no RD do IT-IdP) o que resultaria em nova execução do protocolo *OpenID*.

Para obter valores de Client Response Time, definimos experimentos envolvendo mil casos de logins (autenticações de usuários) e considerando as configurações da camada Secret Sharing como (6,3), (9,6) e (12,10). Nestes mil casos, introduzimos aleatoriamente em 15% dos *logins* a ação de um *OpenID* proxy malicioso. A Figura 29 ilustra os valores de Client Response Time.

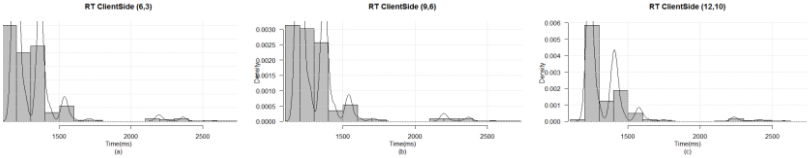


Figura 29 – *Client Response Time* (Com intrusões)

É possível notar que nesta figura que nos casos de autenticações envolvendo proxies maliciosos, os valores de *Client Response* Time são altamente impactados, ficando em todas as configurações em torno de 2,5 segundos. Os *logins* que não tiveram proxies comprometidos mantiveram seus valores de *Client Response* Time em torno de 250 a 300 ms. Este aumento de tempo decorre do fato da necessidade do *SP* detectar problema do proxy malicioso através de um timeout e também de ter que estabelecer uma associação segura com o *proxy backup*. Após este o processo de reconfiguração do protocolo (renovação dos *IdP* proxy), o protocolo volta a funcionar em seu tempo de resposta com valores normais.

5.5.6. Considerações sobre os Testes Realizados

Vale a pena ressaltar que, para o nosso modelo, somente o *Querying Time* tem um peso importante nas avaliações de desempenho. Isto se deve ao fato de que sistemas de autenticação utilizam muito mais leituras (processos de autenticação) do que escritas (criações ou alterações de registros de usuários). Desta forma, ainda que os valores de Register Time sejam maiores que os do *Querying Time*, por serem menos frequentes estas criações ou alterações de contas não influenciam no desempenho do sistema de autenticação.

O temporizador que o SP usa para detectar intrusões no nosso modelo tem um impacto muito grande em execuções de autenticações onde ocorrem a presença de proxy malicioso. Porém os valores de timeout neste temporizador não podem ser assumidos muito pequenos porque provocariam falsos positivos, ou seja, o decurso de prazo definiria reconfigurações de *OpenID* proxies sem a presença real de réplicas maliciosas. Estes valores de *timeout* devem que respeitar principalmente as grandes variações de Querying Time. Em nossos testes, usamos o

timeout de 1 segundo, atendendo com folga desta forma as autenticações sem réplicas maliciosas e sem cache, cujos valores de *Client Response Time* apresentam variações entre 150 e 230 *ms* para as diferentes configurações da camada Secret Sharing. Com intrusões e uma réplica de proxy corrompida, o Client Response Time fica em torno de 2,5 segundos.

Outros experimentos realizados e não apresentados como gráficos foram os testes onde verificamos o efeito de intrusões sobre o algoritmo de compartilhamento de segredos. Nestes casos provedores de cloud poderiam ter armazenadas partes corrompidas devido a ações maliciosas de intrusos. Diante deste tipo de situação o algoritmo de compartilhamento de segredo deve recuperar t partes corretas entre as *n* partes do segredo. Fizemos estas simulações sempre assumindo que as partes corrompidas não podem ultrapassar o limite dado por n – t. A execução do algoritmo, mesmo nestas condições, não mostrou um forte impacto nos valores do *Querying Time* e muito menos no *Client Response Time*.

5.6. Trabalhos Relacionados

Nesta seção, alguns trabalhos citados são confrontados ou mesmo usados como mecanismo de apoio no sentido de evidenciar resultados de nossa proposta. Mas é bom evidenciar que até o momento não temos conhecimento de experiência similar a nossa.

Vários trabalhos na literatura tratam de atributos de usuários e autenticação em *clouds*. Mas, poucos destes coincidem nos seus objetivos com as preocupações que orientaram nossos trabalhos. O projeto SPICE (CHOW et al., 2012) descreve experiências com o objetivo de proteger a identidade de usuários que utilizam uma diversidade de serviços em *clouds*. A preocupação principal é a privacidade dos usuários. No artigo, os autores descrevem que diferentes provedores de *cloud* podem exigir diferentes atributos de usuários. Diante desta constatação, os autores propõem um *framework* com o objetivo de tornar a autenticação do usuário própria para cada provedor de serviço, mantendo a identidade do usuário protegida e liberando apenas os atributos necessários. Ou seja, para demandas de serviço em cada *cloud*, o usuário manterá uma identidade e atributos adequados. Para isto, os autores propõem a criação para cada usuário, um grupo de assinaturas (BONEH; BOYEN; SHACHAM, 2004) que são usadas para autenticação do usuário nos diferentes provedores de *cloud*. Cada assinatura gerada provê os atributos necessários para um determinado provedor. Estas assinaturas são assinadas pelo *framework* SPICE para garantir a validade das mesmas. Outro *framework* semelhante, utilizando criptografia de grupo também é descrito em (YAN; RONG; ZHAO, 2009).

A nossa abordagem embora proteja os atributos de usuários, possui um foco diferente. No nosso caso, as *clouds* estão associadas em federação para fornecer serviços e recursos aos seus usuários. A autenticação do usuário, na nossa proposta, é feita no seu *IdP* de registro e é aceita em todas as *clouds* da federação. As abordagens citadas ((CHOW et al., 2012), (NUÑEZ; AGUDO, 2014) e (BERTINO et al., 2009)), também como a nossa abordagem, tentam evitar a disponibilidade de atributos do usuário nas *clouds*. Mas a ação dos mecanismos destes trabalhos citados está limitada às interações dos usuários no processo de autenticação nas *clouds*. Nós, em nossa abordagem, estamos propondo o armazenamento seguro destas informações em *clouds* da federação. E a autenticação em nosso sistema envolve provedores de identidades que tratam somente da autenticação de usuários em *clouds* desta federação.

A literatura apresenta inúmeras experiências na construção de memória segura em *clouds* ou em federação de *clouds*. Estes trabalhos fazem uso de esquemas de compartilhamento de segredo e de algoritmos de dispersão de informação (*IDA – Information Dispersal Algorithms* (RABIN, 1989) para a memorização segura em *clouds*. Nestas abordagens, a confidencialidade das informações é conseguida com a encriptação das mesmas, usando uma chave simétrica gerada aleatoriamente. Em seguida, o arquivo cifrado é subdividido em partes por uma transformação *IDA*. Por sua vez, a chave criptográfica usada na proteção da informação passa por um processo de compartilhamento de segredo, gerando também partes relacionadas com a chave. As partes do arquivo e da chave simétrica são distribuídas entre várias *clouds*. Esta distribuição deve evitar que tanto as partes do arquivo como as partes da chave venham a ser enviadas aos mesmos provedores de modo a comprometer os seus segredos. A nossa abordagem, é mais econômica no uso destas técnicas. Como os atributos e credencias de usuários correspondendo a registros de poucos *bytes*, na nossa abordagem, podemos aplicar diretamente o compartilhamento de segredo sobre estas informações, evitando com isto o uso de técnicas de correção de erro (transformações *IDA*) e de criptografia para a confidencialidade destas informações.

Em relação ao modelo *IT-IdP* apresentado neste capítulo, acreditamos ser uma experiência única neste nível. Possíveis intrusões resultariam em comprometimento de *IdP*s e dos sistemas que dependem de suas autenticações. No nosso esquema, a invasão de *IdP*s (*proxies* no nosso caso), não resultam em quebra de confidencialidade de dados de usuários e muito menos na interrupção da disponibilidade dos serviços de autenticação de usuários.

5.7. Conclusão

Este capítulo descreveu a nossa experiência no desenvolvimento de provedor de identidades tolerante a intrusões. A arquitetura que implementamos faz uso da tecnologia de virtualização e não envolve o uso de *hardware* especial ou replicação física nos *IdP*s. O modelo apresentado para estes provedores possui duas vantagens principais: a primeira é o isolamento do servidor de autenticação e das outras partes importantes para a correção dos serviços de autenticação em relação aos serviços de rede; e a outra, é a possibilidade de reconfiguração dos *OpenID* proxies tantas vezes quanto for necessário diante dos ataques sofridos por estas partes externas do modelo. Nesta abordagem foram também tomados cuidados para diminuir as possibilidades de *DoS*.

Na elaboração dos protocolos de autenticação, foram tomados cuidados para que as normas usadas fossem respeitadas, em especial as especificações *OpenID*. Especificamente, mantivemos a propriedade que *SPs* compatíveis com *OpenID* deve trabalhar com as nossas proposições sem modificação. O modelo apresentado permite uma certa flexibilidade no acesso às informações, permitindo usuários possam fazer os seus logins a partir de um *IdP* de uma federação de *clouds* e ter acessos a recursos em diferentes partes desta federação.

Também neste capítulo apresentamos mecanismos que permitem o uso de provedores de *cloud* para armazenar as informações de usuário (credenciais e atributos) de forma segura e sempre disponível. Para tanto, aplicamos técnicas de compartilhamento de segredo nas informações de usuário, permitindo que a distribuição de partes destes segredos em várias *clouds* de uma federação. A abordagem de gerenciamento de identidades segue o modelo centralizado devido ao uso de protocolos do *OpenID*. Mas pela flexibilidade proporcionada pela disponibilidade das informações nas *clouds*, temos as mesmas características do gerenciamento de identidades federadas, porém sem a necessidade de relações de confiança entre *IdP*s do sistema.

Um protótipo foi também desenvolvido. Vários testes de desempenho com ou sem intrusões foram realizados neste protótipo e nos dão a certeza da viabilidade das nossas propostas.

O modelo de autenticação apresentado não se restringe a ambientes de federação de *clouds*. Qualquer *IdP* pode ser concebido usando a mesma abordagem e fazer uso de alguns provedores de *cloud* que não formem associações para o compartilhamento de recursos como as citadas federações.

6 CONCLUSÃO

Esta tese descreveu trabalhos que foram desenvolvidos dentro de um contexto de federações de *clouds*. Associações estas que podem ser movidas pela necessidade de pequenos provedores se unirem no sentido de competirem em mercado agressivo. O nosso enfoque sempre foi focado em aspectos de segurança nestes grandes sistemas.

Em um primeiro momento, desenvolvemos uma proposta de federação de *clouds*. Os elementos da organização proposta foram fortemente calcados na nossa experiência em grades computacionais e também na literatura sobre estas associações de clouds. Esta proposta serviu para uma compreensão sobre a ação dos controles de segurança na dinâmica de alocação de recursos nestes grandes sistemas. Neste modelo de federação de *cloud* acreditamos que introduzimos também algumas inovações como a possibilidade de uma espécie de "licitação" onde o cliente apresenta através de seu *broker* suas necessidades e provedores de *clouds* apresentam suas ofertas. Além disso, foram desenvolvidos mecanismos que podem apresentar melhorias na busca de recursos em clouds quando se tratando de federação.

O passo seguinte foram os estudos sobre os controles de autenticação nestes sistemas. Usando nosso conhecimento de gerenciamento de identidades e sabendo que a segurança em suas políticas é fortemente fundamentada na autenticação, partimos para o desenvolvimento de um sistema de autenticação para federações de clouds. Estes trabalhos preliminares de autenticação foram incorporados no projeto *SecFuNet* e adotaram algumas características definidas pelos parceiros de projeto, entre eles, o uso de processadores seguros. Adotamos um modelo de gerenciamento centralizado e a infraestrutura do *OpenID* foi adotada pela sua simplicidade.

Por se tornar ator principal na aplicação de políticas nestas federações, exploramos técnicas que agregassem segurança e tolerância a intrusões nos provedores de identidades que executam os controles de autenticação. Exploramos o uso da virtualização para permitir a proteção de entidades importantes para a segurança destes *IdPs*. A flexibilidade da virtualização permitiu também o uso de esquemas de tolerância a intrusões.

O modelo apresentado para estes *IdPs* possui duas vantagens principais: a primeira é o isolamento do serviço de autenticação e das outras partes importantes para a correção dos serviços de autenticação em relação aos acessos via rede; e a outra é a possibilidade de reconfiguração dos *OpenID* proxies tantas vezes quanto for necessário diante dos ataques sofridos por estas partes externas do modelo. Nesta abordagem foram

também desenvolvidos mecanismos para diminuir as possibilidades de ataques do tipo *DoS*.

Posteriormente, diante de limitações de flexibilidade impostas pelo hardware especializado usado no projeto *SecFuNet*, partimos para uma outra abordagem envolvendo o armazenamento de informações de usuários. Nossa proposta do *IT-IdP*, deixou de estar fundamentada nos princípios deste *hardware* especializado e do uso de *smartcards* e assumiu características mais amplas. Usamos a própria federação de *clouds* para o armazenamento de credenciais e atributos de usuários. No nosso modelo de *IdP*, foram introduzidos protocolos de disseminação baseados em técnicas de compartilhamento de segredo para o armazenamento destas informações nestes ambientes gerenciados por terceiros. Estes protocolos foram desenvolvidos com características de tolerância a intrusões. A flexibilidade proporcionada pela disponibilidade das informações de usuários nas *clouds* faz com que o nosso *IT-IdP*, embora calcado no modelo, apresente as mesmas características do gerenciamento de identidades federadas, porém sem a necessidade de relações de confiança entre *IdPs* do sistema. As técnicas usadas no armazenamento destas informações de usuário garantem as mesmas sempre disponíveis e seguras.

Na elaboração dos protocolos de autenticação, foram também tomados cuidados para que as normas usadas fossem respeitadas, em especial as especificações *OpenID*. Especificamente, mantivemos a propriedade que SPs compatíveis com *OpenID* devem trabalhar com as nossas proposições sem modificação. O modelo apresentado permite uma certa flexibilidade no acesso às informações, permitindo que usuários possam fazer os seus logins a partir de um IdP de uma federação de *clouds* e ter acessos a recursos em diferentes partes desta federação.

A evolução de nossos trabalhos envolveu protótipos. Todos estes protótipos foram submetidos a vários experimentos de testes onde os pontos centrais eram o desempenho e a segurança dos mesmos. A abrangência destes testes nos dá algum conforto sobre a certeza da viabilidade das nossas propostas

6.1 Revisão dos Objetivos e Resultados

Nesta seção revisamos os objetivos propostos inicialmente. O objetivo principal era o estudo dos controles de segurança federações de *clouds*. A autenticação, por ser peça fundamental na implantação de políticas de segurança, acabou sendo o foco principal de nossos esforços. Buscarmos fundamentar nossos esforços em experiências práticas que nos dessem resultados bem próximos do que se experimenta nestes grandes sistemas.

Diante disto, um sistema de autenticação próprio para associações de *clouds* foi desenvolvido. Estes serviços de autenticação foram concebidos tomando como base provedores de identidades que mantêm seus funcionamentos dentro do usual, mesmo na presença de ações maliciosas. São concebidos para suportarem intrusões que corrompem componentes, implantando comportamentos maliciosos. Além disso, por tratar com informações de grande importância, mecanismos de segurança são adotados contra a revelação de dados de usuários.

Para atingir os objetivos iniciais, no seu processo evolutivo nossos trabalhos foram divididos em três partes:

a) Proposta de um modelo de federação de *clouds*.

O modelo de associação de provedores de *cloud* proposta é fundamentado em *broker* que em nome dos usuários negociam recursos para atender suas necessidades computacionais. Os recursos são negociados na forma de uma licitação controlada pelo *broker* do cliente. Com isto, a busca e aquisição de recursos propostas para federações de *clouds* apresentam melhoras consideráveis, tanto ao evitar a contenção dos recursos, assim como na diminuição nas trocas de mensagens necessárias para a negociação destes recursos.

O modelo proposto de federação de *clouds* possui os controles de segurança fundamentados em *IdPs* da própria federação. Para os usuários, esta facilidade permite manter suas credenciais e atributos em um provedor único, não havendo a necessidade de manter diversas contas nos provedores de *clouds* para acessar seus recursos. Além disso, neste modelo de federação, o usuário uma vez que autenticado, permite que o seu *broker* de posse do *token* de autenticação do usuário negocie em seu nome e que os respectivos controles de autorização sejam verificados.

Com base neste modelo de federação de *clouds* desenvolvido e testado tivemos as seguintes publicações:

- BARRETO, L.; FRAGA, J.S.; SIQUEIRA, F.: **Federação de Clouds e Atributos de Segurança**. XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'2015), Vitória, ES. Maio de 2015.
- BARRETO, L.; FRAGA, J.S.; SIQUEIRA, F.: **Architectural Model and Security Mechanisms for Cloud Federations**. The 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (Trustcom'2015), Helsinki, Finland. August 20-22, 2015.

- BARRETO, L.; FRAGA, J.S.; SIQUEIRA, F.: **Conceptual Model of Brokering and Authentication in Cloud Federations**. 4th IEEE International Conference on Cloud Networking (CloudNet'2015), Niagara Falls - Canada. October 5-7, 2015.

b) Proposta de *IdP* tolerante a intrusão com processador seguro.

Exploramos conceitos e ferramentas de forma a aplicar mecanismos de tolerância a intrusões neste tipo de serviço. Este modelo de IdP explora a virtualização para tolerar ações maliciosas e ao mesmo tempo isola componentes e informações importantes da rede. O modelo neste primeiro momento foi desenvolvido sobre uma grade de processadores seguros;

Os trabalhos do *IT-IdP* foram desenvolvidos e testados no quadro do projeto *SecFuNet* e tivemos referente a este trabalho as seguintes publicações:

- BARRETO, L.; SIQUEIRA, F.; FRAGA, J.S. ; FEITOSA, E. **Gerenciamento de Identidades Tolerante a Intrusões**. XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'2013), pg. 805-818. Brasília, DF. Maio de 2013.
- BARRETO, L.; SIQUEIRA, F. ; FRAGA, J.S. ; FEITOSA, E.: **An Intrusion Tolerant Identity Management Infrastructure for Cloud Computing Services**. 20th International Conference on Web Services, 2013, Santa Clara Marriott. International Conference on Web Services, 2013. p. 155-163.
- BOGER, D. ; BARRETO, L. ; FRAGA, J.S. ; SANTOS, A. ; FRANCA, D.T.; Infraestrutura **de Autenticação e Autorização Baseada em SmartCards com Controle de Atributos Centrado no Usuário**. XIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg), Belém, Pará. 11-14 de Novembro de 2013.
- BOGER, D. ; BARRETO, L. ; FRAGA, J.S. ; URIEN, P.; AISSAOUI, H.; SANTOS, A.; PUJOLLE, G.; **UserCentric Identity Management Based on Secure Elements**. 19th IEEE Symposium on Computers and Communications (ISCC´2014), Madeira, Portugal. August, 2014.

c) Armazenamento de credenciais e atributos de usuários em federação de *clouds*.

A análise inicial do *IdP* com *hardware* especializado nos permitiu verificar sua viabilidade, porém, por se tratar de um modelo que é baseado em um único servidor, este modelo poderia se tornar proibitivo em grandes sistemas. As informações de registro de usuários necessitavam ser feitas *off-line* nestes processadores. Isto determinava numa perda de flexibilidade e também de escala. Desta forma, estendemos o modelo do *IT-IdP* para que o mesmo apresentasse as mesmas características de segurança na execução dos protocolos, mas também incrementasse a segurança na manutenção dos dados de usuários sem a necessidade de processadores seguros. Com isto a nossa proposta de *IT-IdP* é incrementada com protocolos que permitem o armazenamento das informações de usuários em recursos das próprias clouds da federação. Técnicas de compartilhamento de segredo foram empregadas para tolerar falhas e ações maliciosas e ao mesmo tempo manter a disponibilidade sem uma replicação extensiva das informações.

Os trabalhos do *IT-IdP* foram desenvolvidos e testados obtendo as seguintes publicações:

- BARRETO, L.; SIQUEIRA, F. ; FRAGA, J.S. ; **Armazenamento Seguro de Credenciais e Atributos de Usuários em Federação de Clouds**. XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'2016), Niterói, RJ, 10-17 Novembro de 2016.
- BARRETO, L.; SCHEUNEMANN, L.; FRAGA, J.S.; SIQUEIRA, F. **Secure Storage of User Credentials and Attributes in Federation of Clouds.** The 32nd ACM Symposium on Applied Computing. (ACM SAC'2017), (**Aceito para publicação**).
- BARRETO, L.; SCHEUNEMANN, L.; FRAGA, J.S.; SIQUEIRA, F. **Securing Identity Providers using Secret Sharing Techniques.** 31th IEEE International Conference on Advanced Information Networking and Applications (AINA'2017), 2017. (**Aceito para publicação**)

Um artigo foi submetido para periódico classificado (*Journal of Information Security and Applications – Elsevier*) e encontra-se em processo de revisão até o momento da escrita destas considerações de conclusão de nossos trabalhos.

6.2 Perspectivas Futuras

Um dos pontos que ainda ficaram em aberto em nosso trabalho é a implementação de forma mais minuciosa do modelo. Apesar dos

resultados apresentados mostrarem a viabilidade da proposta em seus tempos de resposta, percebemos que com o aumento da carga do sistema (autenticações de usuários) o desempenho do sistema começa a apresentar certa instabilidade. Em análise preliminar verificamos que a implementação apresenta falhas em suas estruturas de dados, o que causa esta instabilidade. A melhor implementação do modelo pode ajustar estas instabilidades verificadas em nossos protótipos.

No desenvolvimento de nosso modelo de federação de *clouds* percebemos que o Painel de Recursos foi uma funcionalidade adicional que chamou a atenção por sua eficácia mesmo sendo algo simples. Porém, muitos pontos podem ser explorados sobre este painel, como por exemplo, a escolha dos melhores provedores para atenderem uma demanda, o tempo de espera de resposta, dentre outros. Neste sentido, o Painel de Recursos da federação pode ser também um trabalho a ser estendido futuramente.

A viabilidade da proposta aplicada aos modelos de autenticação nos fez perceber que o tema tem relevância na comunidade científica. Não tivemos um mesmo apuro nos controles de autorização. Neste sentido, para complementar nossos trabalhos, um estudo mais aprofundado sobre as técnicas de autorização para estes ambientes associativos seria interessante.

Na parte relacionada aos algoritmos de compartilhamento de segredos, fizemos usos das abordagens mais conhecidas da literatura, porém novas abordagens que agregam e segurança e desempenho podem ser também encontradas. As aplicações destas novas abordagens também passam a ser um trabalho futuro.

7 REFERÊNCIAS

APACHE, F. **Apache Cassandra**. Disponível em: <http://cassandra.apache.org/>.

APACHE, F. **Apache CouchDB**. Disponível em: <http://couchdb.apache.org/>.

APACHE, F. **Apache JMeter**. Disponível em: <http://jmeter.apache.org/>.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n. 1, p. 11–33, jan. 2004.

BARRETO, L. et al. **An Intrusion Tolerant Identity Management Infrastructure for Cloud Computing Services**. 2013 IEEE International Conference On Web Services. **Anais**...2013

BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. **[RFC 3986] Uniform Resource Identifier (URI)**, 2005.

BERTINO, E. et al. Privacy-preserving Digital Identity Management for Cloud Computing. **Identity**, v. 32, p. 1–7, 2009.

BESSANI, A. et al. DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds. **European Systems Conference (EuroSys'11)**, 2011.

BESSANI, A. N. **JSS - Java Secret Sharing**, 2006. Disponível em: <http://user.das.ufsc.br/~neves/jitt/jss.html>

BHARGAV-SPANTZEL, A. et al. User centricity: a taxonomy and open issues. **Journal of Computer Security**, v. 15, p. 493–527, 2007.

BOGER, D. et al. User-centric Identity Management based on secure elements. **Proceedings - International Symposium on Computers and Communications**, n. 590047, 2014.

BONEH, D.; BOYEN, X.; SHACHAM, H. Short Group Signatures. **Advances in Cryptology - CRYPTO 2004**, v. 3152, p. 227–242, 2004.

BRINKLEY, D. L.; SCHELL, R. R. Concepts and terminology for computer security. **Information Security: An Integrated Collection of**

Essays, n. IEEE Computer Society Press, p. 98–110, 1995.

BURR, W. E.; DODSON, D. F.; POLK, W. T. Electronic authentication guideline. **NIST Special Publication**, v. 800:63, 2006.

BUYYA, R.; RANJAN, R.; CALHEIROS, R. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. **International Conference on Algorithms and Architectures for Parallel Processing**, p. 13–31, 2010.

CALHEIROS, R. N. et al. CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. n. August 2010, p. 23–50, 2011.

CAMENISCH, J.; PFITZMANN, B. Federated Identity Management. **Security, Privacy, and Trust in Modern Data Management**, p. 213–238, 2007.

CARLINI, E. et al. Cloud Federations in Contrail. **Euro-Par 2011 Workshops**, p. 159–168, 2012.

CASTRO, M.; LISKOV, B. Practical byzantine fault tolerance and proactive recovery. **ACM Transactions on Computer Systems**, v. 20, n. 4, p. 398–461, nov. 2002.

CELESTI, A. et al. **How to Enhance Cloud Architectures to Enable Cross-Federation**. Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on. **Anais**...2010a

CELESTI, A. et al. **Three-Phase Cross-Cloud Federation Model: The Cloud SSO Authentication**. Advances in Future Internet (AFIN), 2010 Second International Conference on. **Anais**...2010b

CHADWICK, D. W. Federated identity management. **Foundations of Security Analysis and Design V**, p. 96–120, 2009.

CHAPPELL, D. **Introducing Windows Cardspace**. Msnd technical articles. **Anais**...2006

CHOW, S. et al. Spice–simple privacy-preserving identity-management for cloud environment. **Applied Cryptography and Network …**, p. 526–543, 2012.

CLOUDSTACK, A. F. **Cloudstack**. Disponível em: <https://cloudstack.apache.org>.

COPPOLA, M. et al. The Contrail approach to cloud federations. **The Internation Symposium on Grids and Clouds (ISGC) 2012**, 2012.

D. EASTLAKE; JONES, P. **[RFC 3174] US Secure Hash Algorithm 1 (SHA1)**, 2001.

E. RESCORLA. **[RFC 2631] Diffie-Hellman Key Agreement Method**, 1999.

ECLIPSEFOUNDATION. **Higgins Open Source Identity Framework**, 2010. Disponível em: <http://www.eclipse.org/higgins/>

ELLISON, C. **[RFC 269] SPKI requirements**United StatesThe Internet Society, , 1999.

GOGUEN, J. A.; MESEGUER, J. **Security policies and security models**. 2012 IEEE Symposium on Security and Privacy. **Anais**...IEEE Computer Society, 1982

GOLDREICH, O.; MICALI, S.; WIGDERSON, A. **How to Play ANY Mental Game**. Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. **Anais**...: STOC '87.New York, NY, USA: ACM, 1987Disponível em: <http://doi.acm.org/10.1145/28395.28420>

GROZEV, N.; BUYYA, R. Inter-Cloud architectures and application brokering: taxonomy and survey. **Software: Practice and Experience**, p. 1–22, 2012.

H. KRAWCZYK; BELLARE, M.; CANETTI, R. **[RFC 2104] HMAC: Keyed-Hashing for Message Authentication**, 1997.

HOUSLEY, R. et al. **Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile**United StatesRFC Editor, , 2002.

JABBER SOFTWARE, F. **[RFC 3921] Extensible Messaging and Presence Protocol (XMPP)**, 2004.

JANSEN, B. et al. Architecting dependable and secure systems using

virtualization. **Architecting Dependable Systems V**, n. 1, p. 124–149, 2008.

JIN, X.; KRISHNAN, R.; SANDHU, R. A unified attribute-based access control model covering DAC, MAC and RBAC. **XXVI Data and Applications Security and Privacy**, p. 41–55, 2012.

JOIDS. **JOIDS - Java OpenID Server**. Disponível em: <http://code.google.com/p/openid-server/>.

JØSANG, A. et al. Trust requirements in identity management. **CRPIT '44: Proceedings of the 2005 Australasian workshop on Grid computing and e-research**, p. 99–108, 2005.

JØSANG, A.; POPE, S. User centric identity management. **AusCERT Asia Pacific Information Technology Security Conference**, 2005.

JULA, A.; SUNDARARAJAN, E.; OTHMAN, Z. Cloud computing service composition: A systematic literature review. **Expert Systems with Applications**, v. 41, n. 8, p. 3809–3824, jun. 2014.

KANDALA, S.; SANDHU, R.; BHAMIDIPATI, V. **An Attribute Based Framework for Risk-Adaptive Access Control Models**. Availability, Reliability and Security (ARES), 2011 Sixth International Conference on. **Anais**...2011

KERTESZ, A. Characterizing Cloud Federation Approaches. **Computer Communications and Networks**, p. pp 277-296, 2014.

KHAN, A. R. Access control in cloud computing environment. v. 7, n. 5, p. 613–615, 2012.

KOTLA, R. et al. Zyzzyva: speculative byzantine Fault Tolerance. **ACM SIGOPS Operating Systems Review**, v. 41, n. 6, p. 45–58, 2007.

KUMAR, A. et al. Secure storage and access of data in cloud computing. **2012 International Conference on ICT Convergence (ICTC)**, p. 336–339, out. 2012.

KURZE, T. et al. Cloud federation. **International conference on Cloud Computing, Grids and Virtualization**, n. c, p. 32–38, 2011.

LAMPSON, B. W. **Protection**. Proc. 5th Princeton Conf. on Information Sciences and Systems. **Anais**...1971

LANDWEHR, C. E. Best available technologies for computer security. **IEEE Computer**, p. 86–100, 1983.

LANDWEHR, C. E. Computer security. **International Journal of Information Security**, v. 1, n. July, p. 3–13, 2001.

LANDWEHR, C. E. Engineered Controls for Dealing with Big Data. In: **Privacy, Big Data, and the Public Good: Frameworks for Engagement**. [s.l: s.n.].

LAU, J.; BARRETO, L.; FRAGA, J. Infraestrutura Baseada em Virtualização para Serviços Tolerantes a Intrusões. **XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, 2012a.

LAU, J.; BARRETO, L.; FRAGA, J. DA S. An infrastructure based in virtualization for intrusion tolerant services. **2012 IEEE International Conference on Web Services 2012**, p. 170–177, 2012b.

LEANDRO, M. A. P. et al. Multi-Tenancy Authorization System with Federated Identity for Cloud-Based Environments Using Shibboleth. **ICN 2012, The Eleventh International Conference on Networks**, n. c, p. 88–93, 2012.

LEWIS, J. A. Authentication 2.0 - new opportunities for online identification. **Technical report**, n. Center for Strategic and International Studies, 2008.

LIBERTY. **Introduction to the Liberty Alliance Identity Architecture.Liberty Alliance**. [s.l: s.n.]. Disponível em: <http://www.projectliberty.org/liberty/specifications__1>.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud ComputingNIST Special Publication**. [s.l: s.n.].

MICROSOFT. **[RFC 6749] The OAuth 2.0 Authorization Framework**, 2012. Disponível em: <http://tools.ietf.org/html/rfc6749>

MILLER, J. **Yadis Specification 1.0**. [s.l: s.n.].

MONGODB, I. **MongoDB**. Disponível em: <https://www.mongodb.org/>.

MORGAN, R. L. B.; SCAVO, T. **Shibboleth Architecture Technical Overview**. [s.l: s.n.]. Disponível em: <http://www.internet2.edu/products-services/trust-identity-middleware/shibboleth/>.

NUÑEZ, D.; AGUDO, I. BlindIdM: A privacy-preserving approach for identity management as a service. **International Journal of Information Security**, v. 13, n. 2, p. 199–215, 2014.

OASIS. **Assertions and Protocols for the SAML 2.0**. [s.l: s.n.]. Disponível em: <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>.

OASIS. **Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0**. [s.l: s.n.]. Disponível em: <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>.

OASIS. **Extensible Resource Identifier (XRI) Resolution Version 2.0**, 2008.

OASIS. **Web Services Federation Language (WS-Federation) Version 1.2**. [s.l: s.n.]. Disponível em: <http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html>.

OPENID, F. **Openid Authentication 2.0**, 2007. Disponível em: <http://openid.net/specs/openid-authentication-2_0.html>

OPENID, F. **OpenID Connect**. Disponível em: <http://openid.net/specs/openid-connect-core-1_0.html>.

OPENSTACK, F. **OpenStack**. Disponível em: <https://www.openstack.org/>.

PARK, J.; SANDHU, R. The UCONABC Usage Control Model. **ACM Trans. Inf. Syst. Secur.**, v. 7, n. 1, p. 128–174, 2004.

R. FIELDING et al. **[RFC 2616] Hypertext Transfer Protocol - HTTP/1.1**, 1999.

RABIN, M. O. Efficient dispersal of information for security, load balancing, and fault tolerance. **Journal of the ACM**, v. 36, n. 2, p. 335–348, 1989.

RAGOUZIS, N. et al. **Security Assertion Markup Language (SAML) V2.0 Technical Overview (OASIS)May**. [s.l: s.n.]. Disponível em: <https://www.oasis-open.org/committees/security/docs/draft-sstc-baker-saml-arch-00.pdf>. Acesso em: 28 maio. 2014.

RECORDON, D.; REED, D. OpenID 2.0: A Platform for User-Centric Identity Management. **Proceedings of the second ACM workshop on Digital identity management**, p. 11–16, 2006.

REDHAT. **Infinispan**. Disponível em: <http://infinispan.org/>.

REISER, H. P. VM-FIT: Supporting Intrusion Tolerance with Virtualization Technology. **Workshop on Recent Advances on Intrusion-Tolerant Systems**, 2007.

ROCHWERGER, B. et al. The Reservoir model and architecture for open federated cloud computing. **IBM Journal of Research and Development**, v. 53, n. 4, p. 4:1-4:11, jul. 2009.

ROCHWERGER, B. et al. Reservoir - When One Cloud Is Not Enough. **IEEE Computer Society**, v. 44, n. 3, p. 44–51, 2011.

SANDHU, R.; SAMARATI, P. Authentication, Access Control, and Audit. **ACM Comput. Surv.**, v. 28, n. 1, p. 241–243, 1996.

SCHNEIDER, F. B. Implementing Fault-Tolerant Approach: A Tutorial Services Using the State Machine. **ACM Computing Surveys**, v. 22, n. 4, p. 299--319, 1990.

SCHOENMAKERS, B. A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting. **Advances in Cryptology (CRYPTO99)**, v. 1666, n. i, p. 148–164, 1999.

SETTE, I. S.; FERRAZ, C. A. G. **Integrando Plataformas de Nuvens a Federações de Identidade**. SBRC 2015. **Anais**...2014

SHAMIR, A. How to share a secret. **Communications of the ACM**, v. 22, n. 11, p. 612–613, 1979.

SHIREY, R. **[RFC 4949] Internet Security Glossary**United StatesRFC Editor, , 2007. Disponível em: <http://tools.ietf.org/html/rfc4949>

SUJANA, B. et al. Secure Framework for Data Storage from Single to Multi clouds in Cloud Networking. **International Journal of Emerging Trends & Technology in Computer Science**, v. 2, n. 2, 2013.

TUSA, F. et al. Federation Between CLEVER Clouds Through SASL/Shibboleth Authentication. **INTERNET 2012, The Fourth International Conference on Evolving Internet**, n. 4, p. 12–17, 2012.

VERÍSSIMO, P. E. et al. Intrusion-tolerant architectures: Concepts and design. **Architecting Dependable Systems**, v. 11583, p. 3–36, 2003.

VILLEGAS, D. et al. Cloud federation in a layered service model. **Journal of Computer and System Sciences**, v. 78, n. 5, p. 1330–1344, set. 2012.

W3.ORG. **XMLHttpRequest**. Disponível em: <http://www.w3.org/TR/XMLHttpRequest/>.

WASON, T. **(Liberty Alliance Project) Liberty ID-FF Architecture Overview**. [s.l: s.n.]. Disponível em: <http://www.projectliberty.org/liberty/content/download/1990/13884/file/liberty-idff-arch-overview-v1.2.pdf>.

WOOD, T. et al. ZZ and the Art of Practical BFT Execution. **Proceedings of the sixth conference on Computer Systems EuroSys '11**, p. 123–138, 2011.

YAN, L.; RONG, C.; ZHAO, G. Strengthen cloud computing security with federal identity management using hierarchical identity-based cryptography. **CloudCom 2009**, p. 167–177, 2009.

ZIMMERMANN, P. R. **The Official PGP User's Guide**. Cambridge, MA, USA: MIT Press, 1995.