ALTAIR OLIVO SANTIN

TEIAS DE FEDERAÇÕES: UMA ABORDAGEM BASEADA EM CADEIAS DE CONFIANÇA PARA AUTENTICAÇÃO, AUTORIZAÇÃO E NAVEGAÇÃO EM SISTEMAS DE LARGA ESCALA

FLORIANÓPOLIS

2004

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

TEIAS DE FEDERAÇÕES: UMA ABORDAGEM BASEADA EM CADEIAS DE CONFIANÇA PARA AUTENTICAÇÃO, AUTORIZAÇÃO E NAVEGAÇÃO EM SISTEMAS DE LARGA ESCALA

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutor em Engenharia Elétrica.

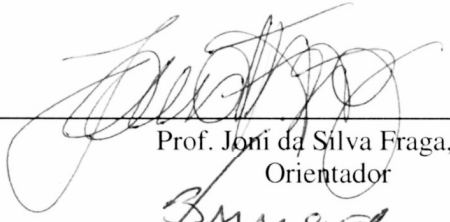
ALTAIR OLIVO SANTIN

Florianópolis, Maio de 2004

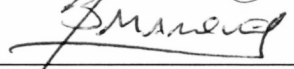
TEIAS DE FEDERAÇÕES: UMA ABORDAGEM BASEADA EM CADEIAS DE CONFIANÇA PARA AUTENTICAÇÃO, AUTORIZAÇÃO E NAVEGAÇÃO EM SISTEMAS DE LARGA ESCALA

Altair Olivo Santin

'Esta Tese foi julgada adequada para obtenção do Título de Doutor em Engenharia Elétrica, Área de Concentração em *Sistemas de Informação*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'



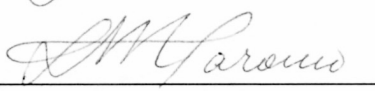
Prof. Joni da Silva Fraga, Dr.
Orientador



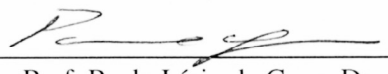
Prof. Jefferson Luiz Brum Marques, Ph. D.
Coordenador do programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

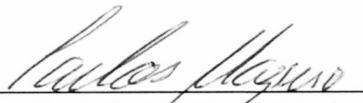
Prof. Joni da Silva Fraga, Dr.
Presidente



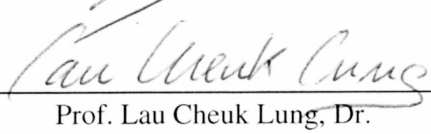
Profa. Liane Margarida Rockenbach Tarouco, Dra.



Prof. Paulo Lício de Geus, Dr.



Prof. Carlos Alberto Maziero, Dr.



Prof. Lau Cheuk Lung, Dr.

Para a mulher que eu amo muito, *Priscila*.

Agradecimentos

Este trabalho contou com a colaboração de várias pessoas, em maior ou menor grau de intensidade, enumerá-las todas seria difícil e eu poderia ser injusto com alguém. Assim, vou citar apenas as pessoas que a minha mente me permite lembrar no momento. Porém, desde já agradeço a todos que direta ou indiretamente contribuíram para que este trabalho fosse possível.

Em primeiro lugar meu especial agradecimento a meu orientador prof. Joni, pelo apoio, compreensão e orientação do trabalho; o prof. Joni sem dúvida contribuiu de maneira definitiva nos momentos mais críticos para que a tese lograsse êxito.

Tenho especial gratidão pelo Emerson pelo seu trabalho nas implementações envolvendo ferramentas relacionadas a este trabalho, pelas várias discussões e pelas sugestões que fez ao trabalho. O Emerson, por sua vez contou com a colaboração dos bolsistas Laura e Diego, a ambos também meus sinceros agradecimentos.

Ao pessoal de apoio e suporte do LCMI, meu agradecimento pela colaboração no desenvolvimento deste trabalho. A Carla, Michelle, Rafael, Eduardo e demais colegas do LCMI agradeço pela oportunidade de poder trocar idéias e enriquecer meu conhecimento e, conseqüentemente melhor os resultados do trabalho. Agradeço também ao Edjard Jamhour pela revisão da tradução.

A minha esposa Priscila agradeço pelas inúmeras revisões do texto, apoio, compreensão e dedicação a mim, mesmo tendo que me dividir com as minhas atividades do doutoramento.

A meus pais, a quem devo minha vida, sou muito grato por tudo o que fizeram por mim sempre que puderam, criando as condições propicias para o meu crescimento em todos os sentidos.

Á Deus agradeço pela oportunidade de poder a cada dia de minha vida ir construindo o meu futuro, agradeço pela saúde que possuo, agradeço pelos “tropeços” que me fazem aprender e não me deixam esquecer que quanto mais sei mais tenho a aprender.

Resumo da tese apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Doutor em Engenharia Elétrica

TEIAS DE FEDERAÇÕES: UMA ABORDAGEM BASEADA EM CADEIAS DE CONFIANÇA PARA AUTENTICAÇÃO, AUTORIZAÇÃO E NAVEGAÇÃO EM SISTEMAS DE LARGA ESCALA

Altair Olivo Santin

Maio/2004

Orientador: prof. Joni da Silva Fraga, Dr.
Área de Concentração: Sistemas de informação
Palavras-chave: Segurança, Controle de acesso, Autenticação e autorização, Cadeias de confiança, Internet
Número de Páginas: 189

Resumo: Os controles de autenticação e autorização clássicos, ditos orientados a nome – onde uma entidade é um nome representando um identificador único no sistema – com uma autoridade de confiança centralizadora, atendem com dificuldades as necessidades dos sistemas de larga escala como as aplicações da Internet. Um novo modelo, dito orientado a chave, mais flexível e escalável vem sendo experimentado na construção de sistemas distribuídos seguros. Neste modelo, é utilizada a infra-estrutura de chave pública sem a centralização da autoridade de confiança; a distribuição da autoridade é feita por delegações sucessivas de permissões numa cadeia de confiança onde o sujeito (beneficiário) é identificado por sua chave pública. A dificuldade nestas cadeias é buscar os caminhos de autorização resultantes das delegações que ligam um cliente a um servidor. As técnicas propostas para buscar estes caminhos, até o momento, não sugerem alternativas quando as buscas falham. Neste trabalho é proposto um esquema para construir tal caminho, com base em uma entidade que representa interesses afins de um agrupamento de principal, a federação. Auxiliar na localização de chaves para resolver nomes e na localização de certificados de autorização para construir os caminhos ligando o cliente ao servidor é a principal função da federação. Para ganhar escalabilidade as federações são agrupadas em teias de federações distribuídas no espaço global. Uma heurística é proposta para que os clientes navegarem na teia de federações buscando certificados. Baseado no middleware CORBA um protótipo do modelo é apresentado, junto a uma avaliação de desempenho do mesmo num ambiente distribuído.

Abstract of thesis presented to UFSC as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering

WEB OF FEDERATIONS: AN APPROACH BASEAD IN TRUST CHAINS FOR AUTHENTICATION, AUTHORISATION AND NAVIGATION IN LATGE SCALE SYSTEMS

Altair Olivo Santin

May/2004

Advisor: prof. Joni da Silva Fraga, Dr.
Area of Concentration: Information Systems
Keywords: Security, Access Control, Authentication and Authorization, Chain of trust, Internet
Number of Pages: 189

Abstract: The traditional authentication and authorization control methods are name-oriented, i.e., each entity is represented by a name that corresponds to a unique identifier in a system. They are, usually based on a centralized trust authority. This approach is barely adequate to fulfill all the requirements associated to a large-scale system, such as Internet. Recently, a key-oriented model has been proposed as an alternative for building secure distributed systems. This model can be considered more scalable and flexible than the traditional ones, because a public key infrastructure is used without a central trust authority. The authority distribution is implemented by successive permission delegations, according to a trust chain, where his/her public key identifies the subject (beneficiary). The difficulty in using the chain approach is to determine (search) the authorization path corresponding to the successive delegations that connect a client to a server. The techniques proposed to determine the delegation chain, until the moment, don't suggest alternatives when the search for a path fails. This work proposes a new strategy to build the delegation chains. The approach introduces a new entity called federation, which represents similar interests of a group of principals. The main function of the federation is to help in the location of keys, for solving names, and authorization certificates, for building the delegation chains. To improve scalability, federations are grouped in "federation webs", distributed along the global network. A heuristic-based algorithm, permitting the clients to navigate across the web federation for searching the certificates, is also presented. A prototype with the main concepts of our strategy was implemented in CORBA. In order to evaluate our proposal, the performance result of this prototype in a distributed scenario is also presented.

SUMÁRIO

LISTAGEM DE TABELAS E FIGURAS

1. Introdução

A segurança (*security*) da informação em ambiente computacional pode ser encarada de várias maneiras pelos seus diversos usuários, por exemplo, como uma necessidade para se resguardar os recursos para um administrador de sistema, um empecilho em suas atividades para um usuário comum ou talvez como um desafio para os intrusos dos meios eletrônico-digitais. Na verdade, segurança é uma disciplina envolvendo técnica, métodos, procedimentos e modelos, tentando se adequar às constantes mudanças tecnológicas para atender as diversas necessidades de seus usuários nos mais diversos tipos de aplicação. A popularização do acesso à rede mundial (Internet) – reunindo os mais diferentes tipos de usuários, objetivos e dispositivos, sendo acessada das mais diferentes localidades – tornou a segurança da informação ainda mais complexa.

É comum fazer segurança num ambiente distribuído, atribuindo a execução de funções de segurança a serviços distribuídos no ambiente que se integram estabelecendo relações de confiança entre si. A maneira clássica de se pensar na efetivação da segurança é a imposição de um conjunto de regras (política) implementadas através de algum tipo de controle; os controles criptográficos e de acesso são os mais comuns. No contexto deste trabalho os controles criptográficos serão utilizados como uma ferramenta para a efetivação dos controles de acesso em suas necessidades e implicações.

Os mecanismos de segurança implementam seus controles geralmente sob a forma de serviços de segurança. Dentre os serviços de maior relevância para este trabalho, estão o

serviço de autenticação e de autorização. O serviço de nomes também será objeto de discussões neste texto. A identificação única implementada através do serviço de nomes é normalmente a base para os controles efetivados nos serviços de autenticação e autorização. Evidentemente, os serviços e os acessos estão condicionados ao tipo de política implementada nos controles de autorização.

De acordo com os propósitos dos sistemas são adotadas políticas para autenticação e autorização, dispostas em serviços distribuídos, configurando então diferentes relações de confiança entre si. O arranjo dos serviços obedece a diferentes técnicas e modelos no intuito de alcançar a segurança desejada, sendo necessário contornar as limitações inerentes a cada tipo de configuração.

1.1 Motivação

A abordagem clássica de concepção de segurança é a centralização da autenticação junto ao serviço de nomes e a distribuição da autorização nos domínios das aplicações. Este modelo, baseado numa entidade centralizadora de confiança, a autenticação, além de propiciar a criação de pontos críticos em relação a vulnerabilidades e falhas, pode impor sérias restrições ao desempenho e a escalabilidade do sistema distribuído em larga escala.

Na abordagem clássica, se considerado o ambiente da rede mundial, a escalabilidade é conseguida através de uma hierarquia global de nomes. Neste caso, as relações de confiança são herdadas nos níveis mais baixos a partir dos níveis mais altos. Porém, o ambiente global é composto por diferentes países autônomos e com legislações próprias, então, como se daria legitimidade a um nome global nos diferentes ambientes locais (países) e como se construiria a hierarquia de países nesta abordagem para estabelecer as relações de confiança?

Os modelos mais difundidos de segurança atualmente seguem a abordagem clássica. Porém, estes modelos na sua maioria fazem uso de mecanismos e técnicas para minimizar os efeitos negativos da centralização, própria desta abordagem. As principais dificuldades nestes modelos são: a escalabilidade limitada e a falta de flexibilidade – indispensável em ambientes distribuídos de larga escala.

As novas tecnologias tanto de hardware quanto de software para uso na Internet vêm generalizando cada vez mais sua utilização e dificultando o emprego dos modelos de

confiança centralizados. Um modelo de segurança mais flexível e de fácil uso, sem nenhuma dependência de entidades centralizadoras, parece mais adequado para estas tecnologias emergentes (ex: dispositivos portáteis, códigos móveis e redes *adhoc*,).

Os modelos que melhor atendem a este tipo de necessidade devem estar baseados numa concepção igualitária onde qualquer entidade possa estabelecer relações de confiança com qualquer outra entidade, sucessivamente, formando o que é denominado de cadeias de confiança. Como a cadeia pode ser global, a maior dificuldade é localizar a cadeia de confiança para um fim específico, dado que não há nenhuma entidade central a qual recorrer para fazer uma busca por permissões/privilégios.

E quando a cadeia de confiança não existe? Como criar esta cadeia em um ambiente distribuído, onde as relações de confiança foram criadas arbitrariamente? Como localizar um principal sem o auxilio de um serviço de nomes? Como localizar direitos de acesso, se as políticas de autorização estão distribuídas pelo sistema distribuído? Depois de localizar o detentor do direito de acesso, como negociar a concessão do mesmo? Como navegar pelas entidades que pertence à cadeia confiança, se a mesma não é estruturada? Que tecnologia deve ser utilizada para construir o protótipo de uma arquitetura que ofereça as características desejadas?

1.2 Descrição do problema e seu contexto

As cadeias de confiança são construídas através da propagação (por delegação) de certificados, denotando a confiança mútua entre cada dois membros da cadeia que constituem os seus elos de ligação. As delegações sucessivas formam cadeias de autorização ou caminhos de confiança. As cadeias são caminhos não controlados, sendo da responsabilidade dos membros das mesmas armazenar e recuperar as seqüências de certificados para resolver nomes e/ou para comprovar autorizações.

Quando um cliente tenta acesso a um servidor de aplicação e não possui uma cadeia de autorização para fazê-lo, não dispõe de nenhum mecanismo para construir tal cadeia. Considerando a existência de um servidor armazenando os certificados de autorização, então seria necessário algum esquema que permitisse negociar a concessão do direito junto ao detentor do mesmo, após localizar o certificado em tal servidor.

Quando se assume um ambiente heterogêneo e distribuído como a Internet esta

negociação se torna uma tarefa difícil. Neste caso, parece haver a necessidade de estabelecimento de relacionamentos de confiança para dar escalabilidade e operacionalizar um sistema integrando vários domínios diferentes.

Como comentado anteriormente, qualquer tipo de configuração que na construção dos relacionamentos de confiança centralize de alguma forma o arranjo das entidades compondo os relacionamentos traz consigo uma série de desvantagens. Por outro lado, uma configuração totalmente descentralizada cria dificuldades de navegação através das entidades do modelo de confiança devido à configuração arbitrária.

Em suma, pode-se dizer que o contexto exige um esquema onde um cliente sem uma cadeia de autorização possa localizar a mesma e negociar a concessão de direitos de acesso em um ambiente totalmente distribuído e com relacionamentos de confiança construído de maneira arbitrária, conseqüentemente sem nenhum tipo de entidade central.

1.3 Objetivos da tese

O objetivo geral deste trabalho é propor um modelo de autenticação e de autorização baseado em cadeias de confiança para sistemas distribuídos de larga escala. O modelo oferece um esquema alternativo para criação de cadeias de autorização ligando um cliente a um servidor. Além disto, pretende-se mostrar a arquitetura e um protótipo do suporte de autenticação e autorização proposto no âmbito das aplicações distribuídas na Internet.

A partir das perspectivas citadas, o projeto de pesquisa perseguiu os seguintes objetivos específicos:

- Desenvolvimento do modelo de confiança para os serviços de autenticação e autorização.
- Criação do mecanismo para localização de cadeias de autorização ligando o cliente ao servidor. Este mecanismo é acionado quando não há caminhos de confiança que levam um cliente a um servidor. O mecanismo permite a localização do detentor do direito e sugere trocas para a negociação da concessão da autorização. Neste caso, o detentor do direito intermedia o processo de criação da cadeia de autorização ligando o cliente ao servidor.
- Desenvolvimento do protótipo implementando o modelo e execução de testes no sentido de validar o protótipo construído.

- Publicação de artigos para relatar os resultados do trabalho em eventos da área.

1.4 Organização do texto

Este capítulo descreveu o contexto geral, a motivação e os objetivos da tese, sendo que os próximos capítulos encontram-se divididos como colocado a seguir.

O capítulo 2 apresenta os fundamentos de um sistema de segurança computacional: políticas, violações, modelos e mecanismos de segurança; a segurança em sistemas distribuídos: a autenticação e a autorização e as várias combinações do controle de autenticação e autorização.

O capítulo 3 discorre sobre a autenticação e autorização orientados a chaves: a gerência de confiança, *SDSI/SPKI* e as experiências com busca de cadeias de certificados.

O capítulo 4 apresenta o modelo de autenticação e autorização proposto: federação, teia de federações, criação de novas cadeias, heurística de navegação pelas teias de federações e um cenário exemplo.

O Capítulo 5 mostra a integração do CORBASec ao SDSI/SPKI: arquitetura do protótipo, implementação do protótipo e aplicação de teste.

O capítulo 6 aborda a avaliação do protótipo através da gerência de risco e faz considerações sobre testes de desempenho do protótipo.

No capítulo 7 são feitas conclusão a cerca dos resultados obtidos com o desenvolvimento da tese.

O trabalho também é composto de um conjunto de anexos versando sobre os seguintes assuntos: Anexo I - Controles Criptográficos, Anexo II - *Kerberos Network Authentication Service*, Anexo III – Infra-estrutura de chave pública, Anexo IV - *Framework de Autenticação X.509*, Anexo V – Arquitetura CORBA (*Common Object Request Broker Architecture*), Anexo VI – Representação e verificação da política de autorização focada na especificação e Anexo VII - Representação e verificação da política de autorização utilizando mecanismos estendidos de autorização.

2. Fundamentos de segurança em ambiente computacional

2.1 Introdução

Nos últimos anos, o interesse por segurança em sistemas computacionais tem crescido muito. Quase sempre, na mídia, quando o assunto é segurança, o foco das notícias se atém a relatos de ações danosas, executadas por invasores[1] e nos grandes prejuízos resultantes da ação. A imagem dos atingidos em tais ações é também fortemente abalada pela repercussão de tais fatos. Se nos ativermos ao citado acima, chegaremos à conclusão que *segurança* se resume a se defender de ataques para evitar prejuízos de alguma natureza que quase sempre geram perdas financeiras.

O que se tenta mostrar neste capítulo é que segurança em sistemas computacionais não é exclusivamente um meio para permear fins, mas é antes de tudo, uma disciplina que através de seus conceitos, metodologias e técnicas, tenta manter as propriedades de um sistema – evitando ações danosas de entidades não autorizadas sobre as informações e os recursos do mesmo. Na literatura, existem várias definições para segurança (*security*) e, em quase todas, é caracterizada a necessidade de se manter no sistema um conjunto de propriedades [1]:

[1] *hackers, crackers, spies e outros tipos de malfeitores.*

- A **confidencialidade** garante a revelação da informação só a sujeitos[2] autorizados.
- A **integridade** assegura a não modificação indevida – seja acidental ou intencionalmente – das informações no sistema.
- A **disponibilidade** garante que as informações e recursos num sistema computacional estarão desimpedidos e prontos para serem usados quando requisitados por sujeitos autorizados.

Outros autores como Landwehr [2] consideram ainda como propriedades de segurança, a autenticidade e o não-repudio.

- A **autenticidade** garante que um sujeito usando uma identificação é seu verdadeiro detentor.
- O **não-repudio** garante que o participante de uma comunicação não possa negá-la posteriormente.

Neste capítulo serão introduzidos um conjunto de conceitos, modelos e técnicas usados no sentido de assegurar as propriedades de segurança da informação. Na seqüência, se abordará a implementação de mecanismos de autenticação e autorização que são as bases fundamentais de segurança em sistemas distribuídos.

2.2 Política de segurança

O termo *política de segurança* pode ter significados diferentes dependendo do nível em que se aplica. Sob a ótica administrativa de uma instituição, por exemplo, pode ser definida como um conjunto de leis e práticas que regulamentam como a instituição gerencia, protege e distribui suas informações. Esta ótica administrativa em relação à segurança da informação deve se refletir também em ambientes computacionais, onde *política de segurança* passa a ser definida como um **conjunto de regras** que especificam como um sistema provê os seus serviços para manter as propriedades de confidencialidade, integridade e disponibilidade. Neste texto, será adotada a noção de política de segurança própria para ambientes computacionais [3].

As políticas de segurança são classificadas em duas categorias: discricionárias e

[2] Entende-se por sujeito uma entidade ativa (programa, processo, sistema, etc.) cujas ações se refletem em recursos do sistema.

obrigatórias. Nas discricionárias os acessos a cada recurso ou informação são manipulados livremente pelo proprietário ou responsável pelo mesmo, segundo a sua vontade (à sua discrição). Nas políticas obrigatórias (não-discricionárias ou mandatórias) as autorizações de acesso são definidas através de um conjunto incontornável de regras, envolvendo a segurança das informações no sistema como um todo [4].

2.3 Violações de segurança

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

As **violações de segurança** em sistemas computacionais correspondem a ação de burlar de alguma forma a política de segurança de modo a não se verificarem uma ou mais das propriedades de segurança. A tabela 2.1 ilustra os tipos de violação em contraposição às propriedades de segurança não verificadas.

Tipo de violação		**Propriedade de segurança violada**
I	revelação não autorizada	confidencialidade
II	modificação não autorizada	integridade
III	negação de serviço	disponibilidade

Tabela 2.1 - Relação de violações as propriedades de segurança

Antes de expor as principais violações em ambiente distribuído, algumas definições devem ser consideradas: ameaça, ataque, vulnerabilidade e *misuse*.

Uma **ameaça** (*threat*) é caracterizada por um conjunto de ações explorando circunstâncias, condições ou conhecimentos sobre um sistema que expõem as propriedades de segurança ao risco (possibilidade de ocorrência de comportamento não esperado do sistema). Uma ameaça, quando posta em ação, é identificada como um **ataque** (*attack*) à segurança do sistema.

Entende-se por **vulnerabilidade** (*vulnerability*) debilidades, fraquezas ou imperfeições em procedimentos, serviços ou sistemas. Estas vulnerabilidades podem ser

oriundas de falhas de concepção, implementação ou de configuração de serviços ou aplicações, expondo os recursos de um sistema computacional a ataques.

É muito comum se encontrar na literatura, também, o termo ***misuse,*** que corresponde ao uso incorreto, impróprio ou excessivo de um recurso.

A tabela 2.2 relaciona as ameaças / ataques mais comumente presentes em relatos de violações (alertas / boletins de segurança) em sistemas distribuídos e rede de computadores.

Ameaça / ataque	**Tipo de violação**	**Descrição**
Mascaramento (*masquerade/spoofing*)	II	Técnica utilizada para "se fazer passar"/forjar uma identidade. Utilizada por exemplo em capturadores de senha, *IP spoofing*, etc.
Bypassing control	I	Técnica utilizada para explorar vulnerabilidades do sistema. É comum a utilização desta técnica em invasões de sistemas com versões (*patchs, service packs*) desatualizadas.
Código maléfico (*malicious code*)	I II III	Técnica que utiliza software com código contendo partes aparentemente inofensivas ou invisíveis. Esta, quando executada, compromete a segurança dos recursos do sistema. Esta técnica é utilizada, por exemplo, em cavalos de tróia, vírus, bombas lógicas, etc.
Backdoor/Trapdoor	I II	Técnica utilizada para inserir/criar funções/escutas no sistema, que aceitam entradas específicas e permitem contornar/driblar os mecanismos de segurança do sistema.
Inspeção de lixeira (m*edia scavenging*)	I	Técnica que consiste em bisbilhotar/revirar lixeiras procurando papéis, discos, etc. com informações importantes que não foram adequadamente destruídas.
Scanning	I	Técnica utilizada para bisbilhotar recursos do sistema com algum objetivo específico (*portscan*, por exemplo). Um ataque deste tipo utiliza-se de um software *scanner* para ser executado.
Negação de Serviço	III	Técnica utilizada para impedir o acesso legítimo ao sistema ou a algum recurso deste. Alguns exemplos desta técnica são *Distributed Denial of Service*, *Worms*, repetidor de discagem e diversas variações de *Denial of Service* (DoS).
Ataque nas Comunicações	I II	Técnica utilizada para escuta, interceptação, inserção ou alteração de mensagens durante a sua transmissão. Geralmente, a escuta da comunicação com um software *sniffers*, por exemplo, precede o uso de ataques como *Man-in-the-middle* ou roubo da sessão autenticada (*hijacking*).

Tabela 2.2 – Relação entre as ameaças / ataques e os tipos de violação (tabela 2.1)

2.4 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 [5]. Os modelos se apresentam na literatura divididos em três tipos básicos [6] [7]: baseados em identidade[3] ou **discricionários** (*discretionary*), baseados em regras ou **obrigatórios** (*mandatory*) e os **baseados em papéis** (*role*).

2.4.1 Modelo discricionário (discretionary)

O modelo discricionário, em sua essência, é baseado no modelo **matriz de acesso** proposta por *Lampson* [8]. Neste modelo, uma matriz relaciona **objetos** (recursos), **sujeitos** (entidade ativa) e **atributos** de acesso (direitos ou permissões de acesso – por exemplo: *read, write, ...*).

A Figura 2.1 mostra a matriz de acesso, onde cada sujeito (S1, S2, S3, etc.) é representado por uma linha e cada objeto (O1, O2, etc.) por uma coluna. Na célula onde ambos (linha x coluna – Figura 2.1) se encontram são especificados os atributos de acesso do respectivo sujeito em relação ao objeto considerado.

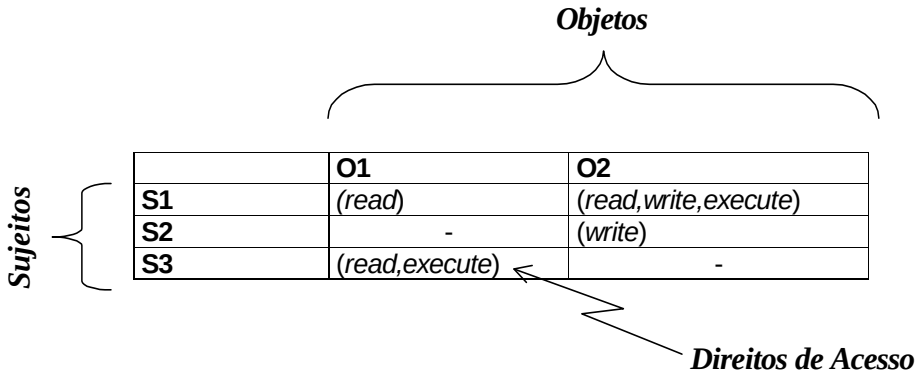


Figura 2.1 - Modelo matriz de acesso

Considerando uma coluna da matriz de acesso, tem-se que a relação de todos os sujeitos com os seus respectivos direitos de acesso sobre o objeto correspondente formam a **lista de controle de acesso** (*Access Control List - ACL*) do objeto considerado. As *ACLs* correspondem a uma forma de apresentação do modelo matriz de acesso. Na Figura 2.2 é

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

apresentado um conjunto de *ACL*s, onde cada entrada corresponde a uma lista de controle de acesso para cada objeto.

Objeto	***Listas de Controle de Acesso***
O1	S1(read), S3(read, execute)
O2	S2(write), S1(*read*, *write*, *execute*)

Figura 2.2 - Listas de Controle de Acesso

A matriz de acesso pode ter uma outra representação através de **listas de competências** (***capabilities*** *list*). Só que neste caso, a *capability* relaciona o(s) objeto(s) e respectivos direitos que o sujeito possui sobre o(s) mesmo(s). Na Figura 2.3, cada entrada é representada numa lista (de *capabilities*) por um sujeito e determina as permissões que este tem sobre os objetos do sistema.

Sujeitos	***Listas de Competências***
S1	O1(read), O2(read, write, execute)
S2	O2(write)
S3	O1(read, execute)

Figura 2.3 - Listas de Competências

2.4.2 Modelos Obrigatórios (mandatory)

Os modelos obrigatórios caracterizam as políticas ditas mandatórias definindo regras e estruturas válidas no âmbito de um sistema, normalmente, especificando algum tipo de política multinível (*multilevel policy*). Políticas multinível estão baseadas em algum tipo de classificação, as quais estão submetidos os acessos dos sujeitos aos objetos [9].

Uma das formas de viabilizar a implementação destas políticas é construir **reticulados** (*lattice*[4]) com **rótulos de segurança** (*security labels*). Os rótulos são constituídos de níveis de sensitividade e categorias (*category*). As categorias correspondem a compartimentos específicos associados a um nível de sensitividade. Os níveis de

[4] Lattice corresponde a um conjunto matemático de elementos ou recursos do sistema parcialmente ordenados (satisfazendo uma relação que é transitiva e antissimétrica), tal que para quaisquer 2 elementos, existe um elemento que é o maior no subconjunto de todos os elementos menores ou iguais a ambos e, um elemento que é o menor no subconjunto de todos os elementos maiores ou iguais a ambos.

sensitividade atribuídos às informações descrevem algum tipo de classificação empregada no sistema.

Os rótulos de segurança correspondem ao produto vetorial entre o conjunto de níveis de sensitividade e o conjunto de *Category* (*Security label = Sensitivity level* x *Category*); sendo que *Category* denota o conjunto de todos os subconjuntos formados a partir das categorias definidas no modelo.

Assim, o *security label* é o conjunto de todos os pares ordenados (a, b) onde a, o primeiro elemento, corresponde ao nível de sensitividade da informação no sistema e b descreve o subconjunto *category* (que pode até ser vazio) a que pertence a informação. Neste contexto o *sensitivity label* do sujeito toma o nome de **autorização** ou **habilitação** (*clearance*). No caso de objetos (entidade passiva no sistema) o *sensitivity label* é denominado nível de **classificação** (*classification*).

Para permitir comparações entre os *security labels* de sujeitos e objetos nestes modelos de políticas multinível, é definida a seguinte **relação de dominância**:

Security level (s_1,c_1), ***domina*** *Security level* (s_2,c_2)
se são verificadas as seguintes condições:
$Sensitivity(s_1) \geq Sensitivity(s_2) \wedge cats\ (c_1) \supseteq cats\ (c_2)$

No caso, *cats(x)* fornece o subconjunto de *category* a que c_x pertence. Os operadores "≥" e "⊇" indicam, respectivamente, ordem entre os níveis de sensitividade e a condição de superconjunto entre os conjuntos que compõem *category*; o operador "^" na condição acima representa agregação ("e").

A seguir são descritos os principais modelos obrigatórios/mandatórios constantes da literatura.

A. O modelo Bell-LaPadula (BLP) de confidencialidade

O modelo *BLP* [10] classifica as informações em 4 níveis hierárquicos de sensitividade (não classificada, confidencial, secreta e ultra-secreta – respectivamente do menor para o maior nível) e num conjunto não hierárquico de categorias ou compartimentos.

Para evitar a revelação da informação a sujeitos não autorizados, basicamente, o modelo *BLP* impõe duas regras:

- ***NRU*** (*No Read Up*) ou **propriedade de segurança simples** (*simple security propriety* ou *ss*): esta regra evita que um sujeito de nível inferior leia informações de um nível mais elevado que o seu nível de segurança (habilitação e compartimento). Ou seja, o nível de sensitividade do sujeito deve dominar o do objeto.
- ***NWD*** (*No Write Down*) ou **propriedade estrela** (**-propriety*): esta regra limita as ações de um sujeito com *security label* x_s, em seus acessos de escrita a objetos com nível de segurança x_o; x_o deve dominar x_s.

A preocupação nestas regras é evitar que a informação de um nível mais alto acabe fluindo para níveis baixos de segurança, caracterizando a revelação não autorizada de informação. Além das duas propriedades descritas o modelo também mantém um controle discricionário por nível de segurança (*discretionary security propriety*) que reflete os princípios de autorização expressos no modelo matriz de acesso.

A dinâmica do *BLP* é descrita por uma máquina de estados onde a transição de um estado seguro (condição em que nenhuma das propriedades de segurança é violada), para outro estado seguro só é possível se as propriedades estrela e simples forem mantidas – no acesso que define a transição.

O modelo de *BLP* apresenta entre suas principais limitações [11] [4] a escrita às cegas e a superclassificação da informação. A **escrita às cegas** (*blind write*) é assim denominada porque a propriedade estrela permite que um sujeito escreva num objeto de nível de sensitividade superior a sua habilitação, podendo ocorrer a destruição de informações sem caracterizar uma violação das regras do modelo propriamente dito. A **superclassificação da informação** é determinada pelas regras do modelo, definindo uma tendência de fluxo da informação dos níveis de segurança mais baixos para os mais altos. Com o passar do tempo a superclassificação dificulta a manipulação da informação e a operacionalização do sistema.

Por exemplo, durante um processo de cópia, quando a classificação do objeto original (que sofrerá a operação de leitura) é inferior a habilitação do sujeito, a propriedade estrela impõe que o objeto copiado (que sofreu a operação de escrita) tenha classificação igual ou superior a habilitação do sujeito. Neste caso, é possível perceber pela situação exposta que a classificação da informação foi “puxada” para cima no objeto copiado. Com o passar do tempo à informação tende a subir nos níveis de classificação a ponto de em

dado momento necessitar uma reclassificação para tornar o sistema usável. Na literatura existem várias técnicas propostas para tratar estes problemas do *BLP* [11] [4].

Os **sujeitos de confiança** (*trusted suject*) foi considerado já em [10], para que processos possam violar a propriedade estrela, se necessário, sem comprometer a segurança do sistema, tratando entre outras coisas a superclassificação da informação.

B. O modelo Biba de integridade

O modelo *Biba* é definido como o dual do *BLP*. Suas regras são similares ao do modelo anterior, porém, tem como objetivo, a preservação da integridade das informações classificadas, evitando modificação não autorizada.

O modelo define níveis hierárquicos de integridade para os sujeitos (I_s) e para os objetos (I_o) similares aos níveis de sensitividade definidos no *BLP*. A **propriedade simples de integridade** define que um sujeito só pode ler um objeto se seu nível de integridade for dominado pelo objeto ($I_o \geq I_s$). A **propriedade estrela de integridade** especifica que sujeitos podem ter direito de escrita sobre objetos, se e somente se $I_s \geq I_o$.

Por ser o dual de *BLP*, este modelo apresenta limitações similares às descritas naquele. No modelo *Biba* ocorre uma degradação do nível de integridade, de maneira análoga a superclassificação da informação do modelo de *BLP*. Via de regra, também há a necessidade de **sujeitos de confiança** no modelo Biba para alterar a integridade de sujeitos e objetos, mantendo o sistema viável.

C. Outros Modelos Mandatórios

Na literatura são identificados outros modelos mandatórios além do *Bell-Lapadula* e do *Biba*. O **modelo *Clark-Wilson*** (*CW*) [12] é baseado na idéia que a integridade é mais importante que a confidencialidade para operações comerciais. Diferentemente dos modelos *Bell-LaPadula* e *Biba*, o *CW* assume **transações bem-formadas** (“todos os passos de uma seqüência de atividades são executados corretamente”) e a **separação de tarefas** (“cada sujeito desempenha um papel distinto na seqüência de atividades que formam uma transação”) como essência de sua definição.

Outra classe de modelos mandatórios presente na literatura é a dos modelos de **controle de fluxo**, introduzidos no sentido de tratar os problemas de confinamento (canais

cobertos temporais, canais de memória, cavalos de tróia, etc.). Nestes modelos não são considerados mais acessos de um sujeito em objetos, mas de fluxos de informação entre sujeitos. Estes modelos tratam de identificar os canais de comunicação legítimos e os ilegítimos. O modelo descrito em [13] é um exemplo desta classe de modelos.

2.4.3 Modelos Baseados em Papéis (Role-Based Access Control - RBAC)

Em modelos RBAC, a idéia central é que o usuário desempenhe diferentes papéis em um sistema. Um **papel** pode ser definido como uma função em uma organização. Assim, no RBAC as permissões são conferidas aos papéis e os usuários são autorizados a exercer papéis (Figura 2.4).

O controle de acesso baseado em papéis facilita a gerência de autorização, porque quando o usuário muda de atribuição – sendo, portanto, desligado de um papel e assumindo um outro – a manutenção das permissões dos papéis não sofre mudanças.

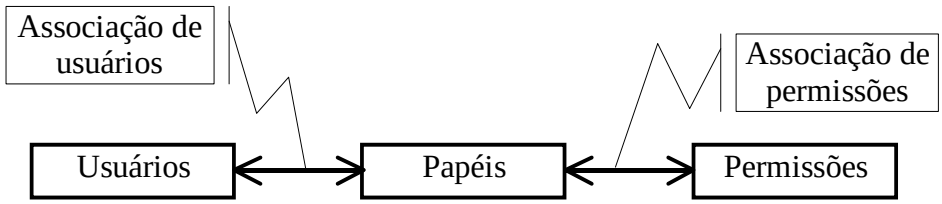


Figura 2.4 - Modelo RBAC Básico

O RBAC não é um conceito novo, mas só recentemente ganhou a atenção dos pesquisadores. Um modelo unificado denominado *RBAC-NIST* foi criado para tentar padronizar as várias tendências que têm surgido em modelos de papéis [14].

Esta família de modelos *RBAC-NIST* se apresenta estratificada a partir do modelo RBAC básico (*Flat RBAC*), evoluindo pelos outros modelos da família – RBAC Hierárquico (*Hierarchical RBAC*), RBAC com Restrições (*Constrained RBAC*) e RBAC Simétrico (*Symmetric RBAC*) [14].

O RBAC **básico** implementa a infra-estrutura de base composta pelo usuário, permissões e papéis com suas respectivas semânticas. A relação entre os mesmos deve ser do tipo muitos-para-muitos (Figura 2.4). Este modelo define funções administrativas, interpretadas como suporte à revisão das associações usuário-papel – o que determina os usuários associados a um papel e vice-versa.

O RBAC **hierárquico** acrescenta ao RBAC básico uma hierarquia que permite estruturar papéis de maneira a refletir a hierarquia de responsabilidades reais de uma organização.

O RBAC **com restrições** está baseado no **princípio do mínimo privilégio** (suporta a *separação de tarefas*: o usuário só utiliza os direitos necessários para uma dada tarefa), impondo desta forma restrições na ativação de papéis conflitantes.

O RBAC **simétrico** inclui funções administrativas que dão suporte à revisão de associações permissão-papel – o que torna possível identificar as permissões associadas a um papel e os papéis que possuem determinadas permissões.

2.4.4 Considerações sobre os modelos

Basicamente os modelos de segurança foram definidos em períodos distintos com tendências e características específicas: no começo dos anos 60 - estimulados pelo desenvolvimento dos sistemas de tempo compartilhado, nos anos 70 - com fins e propósitos militares e, nos anos 80 com enfoque mais comercial. A partir dos anos 90, a tônica vem sendo os modelos que envolvem ambientes distribuídos, sem um propósito específico, mas geralmente para suportar políticas múltiplas.

O modelo matriz de acesso é caracterizado pela sua flexibilidade, o que facilita a gerência descentralizada de direitos. O preço da flexibilidade e da descentralização é a complexidade envolvida no controle a propagação de direitos no sistema.

Os modelos de controle mandatário geralmente definem um conjunto de regras não contornáveis no sistema que diminuem as possibilidades de propagação de direitos. Estes modelos são próprios para gerências centralizadas de segurança, representando estruturas pouco flexíveis.

Os modelos baseados em papéis são tidos como modelos intermediários entre os discricionários e os obrigatórios, apresentam a estrutura flexível dos primeiros e gerência que pode ser centralizada dos obrigatórios. Todos os modelos descritos neste texto podem ser implementados em sistemas distribuídos, com maior ou menor dificuldade [15].

2.5 Mecanismos de segurança

Os mecanismos são responsáveis pela implementação das políticas de segurança, expressas pelos modelos de segurança. Para viabilizar a implantação de tais políticas, os mecanismos são construídos a partir de controles de acesso e controles criptográficos.

No que se refere a controle de acesso esses mecanismos tomam o nome de **monitor de referências** [16], intervindo em vários níveis de um sistema.

O monitor como responsável por intermediar todas as requisições de acesso aos objetos de um sistema deve ter algumas propriedades: deve ser inviolável, incontornável (sempre invocado) e pequeno o suficiente para permitir a verificação de sua correção. A noção de **núcleo de segurança** foi definida em [17] como o conjunto de recursos de hardware e software que permitem a concretização de um monitor de referência.

A implementação do monitor de referência é feita por mecanismos de controle que podem ser discricionários (DAC – *Discretionary Access Control*), obrigatórios (MAC – *Mandatory Access Control*) ou baseados em papéis (*RBAC - Role-Based Access Control*).

2.5.1 Controle de Acesso Discricionário (DAC)

Os mecanismos DAC implementam políticas discricionárias, permitido ao usuário atribuir direitos de acesso aos seus recursos computacionais de acordo com a sua vontade. Porém, se o usuário não atribuir corretamente estes direitos ou mesmo se o fizer permitindo acesso de cópia a outros sujeitos, a disseminação de suas informações pelo sistema pode não ser controlada.

O controle de acesso discricionário não impõe nenhuma restrição à disseminação de direitos e a própria evolução da matriz de acesso. Na prática, devido talvez à facilidade de implementação e flexibilidade, o controle de acesso discricionário é largamente utilizado nos sistemas atuais.

2.5.2 Controle de Acesso Mandatório (MAC)

Em mecanismos MAC, que implementam a política obrigatória, as regras de controle de acesso são impostas por uma autoridade central. Estes mecanismos, como descrito

anteriormente, implementam políticas multinível. Dos vários relatos de implementação de mecanismos MAC, observa-se que os mesmos são bem mais difíceis de viabilizar que os mecanismos de DAC, devido à rigidez de suas regras e as limitações de seus modelos, além de outras dificuldades de natureza computacional [18].

2.5.3 Controle de Acesso Baseado em Papéis (RBAC)

Para os mecanismos que implementam RBAC a identidade no sistema é o papel, uma vez que estes encapsulam as políticas na forma de permissões. Por ser independente das políticas, o *RBAC* é facilmente ajustável a mudanças no ambiente; não é tão flexível quando o DAC e nem tão rígido quanto o MAC. Por mais que os modelos *RBAC* ainda estejam em desenvolvimento, existem vários relatos de implementações do mesmo, um deste pode ser encontrado em [19].

2.5.4 Controles Adicionais de Segurança

Os mecanismos de segurança envolvem ações, técnicas, procedimentos ou dispositivos que têm como propósito implantar políticas de segurança. Outros controles (ditos internos) que não atuam diretamente nas requisições de acesso devem estar presentes nos sistemas:

- **auditoria de vestígio**: ligada à geração periódica de registros de eventos associados à segurança – coletados para uso potencial em detecção de intrusão e/ou auditoria de segurança.
- **auditoria de segurança**: inspeção independente (executada por terceiros) dos procedimentos e registros do sistema com intuito de verificar possíveis violações do sistema e para a adequação das políticas de segurança empregadas.
- **detecção de intrusão**: usa os mesmos registros das auditorias em métodos automatizados de análise em tempo real, envolvendo muitas vezes uma seqüência de eventos relacionados ou não; o intuito é 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 (seção 2.3) não puderam ser evitadas

completam os controles adicionais.

Outros controles (ditos externos) necessitam ser adicionados aos internos já comentados, por exemplo, é necessário que se leve ao conhecimento de cada usuário suas atribuições e responsabilidades (*accountability*), 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 sem eficácia.

2.6 Segurança em sistemas distribuídos

Ao se considerar sistemas distribuídos, a segurança assume características especiais: a preocupação não se resume mais a uma máquina ou recursos geridos de forma centralizada. Os recursos computacionais são dispostos formando redes de computadores, normalmente, ocupando amplas extensões geográficas.

Os sistemas distribuídos são caracterizados pela heterogeneidade de seus componentes, envolvendo diferentes tecnologias. Além disto, são ambientes que apresentam uma dinâmica em suas composições, possuindo números sempre crescentes de recursos, o que pode se refletir em *problemas de escalabilidade*.

É prática comum – no sentido de vencer problemas de escalabilidade nestes ambientes – reunir conjuntos de recursos computacionais destinados a um mesmo fim em um **domínio**. Cada domínio apresenta políticas de autenticação e de autorização específicas garantindo as propriedades de segurança dos recursos no domínio**.**

2.6.1 Autenticação em sistemas distribuídos

O processo clássico de identificação de um principal[5] consiste na sua individualização dentro de um grupo de usuários do sistema a partir de uma **prova de identificação**, que pode ser manifestada através da posse de alguma informação, do conhecimento de algum segredo ou ainda, da posse de uma característica única (íris, impressão digital etc.). A identificação é definida dentro de um espaço de nomes, onde cada identificador é atribuído unicamente a um principal e é válido em todo o sistema.

[5] Entidade ativa que pode provar sua identificação, por exemplo: pessoa, sistema, processo, etc

Em um sistema distribuído − num cenário onde um principal tem acesso a uma máquina *A* e a uma *B*, se o principal que passou pelo processo de identificação no sistema *A* desejar acessar também o sistema *B*, pode-se imaginar duas situações: (i) o principal se autenticando novamente através da rede no sistema *B* ou (ii) o sistema *A* enviando informações de identificação do usuário pela rede ao sistema *B*.

Em ambos os casos (i) e (ii), haveria a necessidade de mecanismos para proteger as informações de identificação em trânsito pela rede (necessidade de controles criptográficos, por exemplo). Além disto, no caso (i), o usuário teria que passar novamente pelo processo de autenticação no sistema *B*, ou seja, enviar novamente senhas e informações de identificação pela rede; este caso se complica quando o número de máquinas necessário para o processamento desejado pelo usuário é relativamente grande. O caso (ii) se apresenta como mais simples, entretanto, os sistemas *A* e *B* deveriam compartilhar algum tipo de "segredo", de tal modo que *B* pudesse confiar em *A* a ponto de ter certeza que as informações de identificação recebidas como oriundas de *A* fossem sempre autênticas.

Para resolver problemas como os do cenário (ii), é necessário uma **relação de confiança** com uma terceira entidade (*trust third party*) – **o serviço de autenticação**.

Tal relação é sustentada no pressuposto que esta entidade autenticadora/certificadora não apresentará comportamento diferente do esperado. Assim, para fins de identificação, o serviço de autenticação é o fornecedor das informações necessárias para a autenticação dos dados sobre o principal. Estas informações constituem o que é normalmente chamado de **certificado**, representando uma prova irrefutável de autenticidade, aceita pelos participantes da comunicação − emissor e destinatário assumem a credibilidade do serviço de autenticação que gerou essas informações de certificação.

No cenário do caso (ii) acima os dados sobre o principal, enviados pelo sitio (*site*) *A* são confrontados com as informações do certificado no destinatário *B* − na verificação da autenticidade dos mesmos.

O uso de certificados evita a necessidade de mecanismos específicos para proteger informações de autenticação em trânsito pela rede, pois os mesmos estão baseados em assinatura digital ou outro mecanismo gerado pelo certificador para garantir a integridade e autenticidade do certificado. Na identificação fica também desnecessário enviar informações confidenciais como senhas pela rede para sítios remotos, pois os certificados

podem ser utilizados para comprovar a autenticidade de um principal, o que viabiliza a identificação única (*single sign-on*).

Como os sistemas distribuídos são construídos para permitir operações remotas que envolvem trocas de mensagens e, pelo exposto acima, percebe-se que nestes sistemas a autenticação não se limita à identificação de principais, é necessária também à autenticação de mensagens em trânsito. Neste caso, se deseja a propriedade de autenticidade das mensagens, através da identificação incontestável da origem e do destino das mesmas, comumente alcançada pelo uso de certificados.

Os certificados presentes em serviços de autenticação usuais − como o *Kerberos* (chave secreta) [20] ou o *X.509* (chave pública) [21] − normalmente estão baseados em mecanismos de assinatura digital. Assim, pode-se pressupor que em sistemas distribuídos a autenticação é suportada por mecanismos que em geral reduzem o processo de autenticação à comprovação da posse de uma chave criptográfica [22] [23] [24] [25] [26].

Na seção 2.7 vai será apresentado um estudo de casos, a disposição deste serviço de autenticação em sistemas distribuídos e a composição dos certificados.

2.6.2 Autorização em sistemas distribuídos

A autorização é um processo onde o objetivo é garantir que só tenha acesso aos recursos os principais com legitimidade para fazê-lo. O processo de autorização consiste de duas fases: a representação (*representation*) e a verificação (*evaluation*).

A **representação** (da política) compreende a expressão dos atributos (*requeriments*) de autorização no sistema distribuído. A maneira mais comum de representação destes atributos (direitos, permissões ou privilégios) é sua expressão através de uma *ACL* ou uma *capability list* (seção 2.4.1).

A atividade de **verificação** do acesso compreende a ação de recuperar os *requeriments* associados a um objeto e confrontá-los com os atributos de autorização associados ao principal na sua tentativa de acesso.

Um principal efetuando acessos a um objeto pode se valer dos atributos que possui ou fazê-lo através de atributos recebidos de terceiros via delegação [27]. A **delegação** se constitui na ação de conceder atributos de autorização a um principal que agirá na efetivação do acesso desejado.

Na literatura de sistemas distribuídos alguns autores têm apresentado abordagens para a representação e a verificação da autorização que estendem a maneira clássica de concretizar a autorização. Uma destas abordagens − partindo do pressuposto que a política deve ser independente dos mecanismos de autorização − é baseada numa preocupação maior com o formalismo da especificação da (política de) autorização [28]. Uma generalização de *ACL* (*GACL*) é utilizada para especificar políticas de maneira mais "modular", similar a linguagens de programação (procedurais), não se limitando apenas a fazê-lo de maneira declarativa, como é feito nas *ACL*s tradicionais.

Para ilustrar como a representação de uma política de autorização é expressa através da *GACL*, considere os usuários *Alice* e *Bob* e os grupos *Dept* e *Research*, que têm seus *requiremets* de autorização para os objetos *P.exe* e *P.src* descritos conforme visto na representação parcial da *GACL* mostrada na Figura 2.5.

Como a representação da política começa em *P.exe* (Figura 2.5), aparece após o *declare* a palavra *ordered*, indicando que a verificação da parte *list* deve ser feita seqüencialmente.

Na assertiva '*<[Alice,Bob],[- execute]>*' (Figura 2.5) é especificado que os usuários *Alice* e *Bob* não possuem permissão de execução no objeto *P.exe*. O contrário é especificado na assertiva seguinte, '<[*Dept*],[*execute]>*', que permite a execução aos membros do grupo *Dep*.

A assertiva '*inherit P.src::<[*],[write*]>' (Figura 2.5) especifica que todos os usuários que têm permissão de escrita em *P.src* também a terão em *P.exe*, no caso, apenas os membros do grupo *Research*.

P.exe	*declare*	*Ordered*
	list	<[Alice,Bob],[-*execute*]>,
		<[*Dept*],[*execute*]>,
		inherit P.src::<[*],[*write*]>
P.src	*declare*	
	list	<[Research],[*read*,*write*]>,
		<[-*Dept*],[-*write*]>

Figura 2.5 - Representação de política de autorização com a GACL

O *framework GACL* também suporta delegação e sugere uma infra-estrutura de servidores para autorização distribuída incluindo operações e protocolos, que não serão abordados neste contexto. De maneira geral, pode-se considerar que tal abordagem tem as

vantagens conhecidas de linguagens de especificação, mas a desvantagem de ter a sua representação limitada a um tipo de mecanismo de autorização, a *ACL*. Maiores detalhes sobre a *GACL* podem ser obtidos no Anexo VI.

A abordagem apresentada em [29] é mais geral e tem como base extensões (*extend*) aos mecanismos de autorização (*ACLs* ou *capabilities lists*) para suporte às restrições de acesso (direitos) que também são expressas através de uma linguagem de autorização. A linguagem introduzida neste caso é constituída por seqüências de *tokens* (estrutura de dados composta de 3 campos), usados na descrição de atributos (*requirements*) da política de autorização. Através dos *tokens* é possível representar de maneira uniforme modelos de autorização baseados em *ACL*s, *capabilities lists* ou reticulados (*lattice-based*).

A Figura 2.6 mostra um exemplo de *EACL* (*Extend Access Control List*) implementada a partir de *tokens*. No exemplo, o usuário *Tom* identificado através da credencial *X.509* (#1) e o grupo *admin* do domínio *kerberos* ORG.EDU (#2) estão autorizados (*pos_access_rights*) a ter acesso (#1 − '*read*' e #2 − '*read* e *write*') ao objeto *FILE*; o objeto *FILE* é de responsabilidade do gerenciador de arquivos (*local_manager*).

EACL entry	*token type*	*def.authority*	*value*
#1	access_id_USER	X.509	"/C=us/O=alliance/CN=Tom"
	pos_access_rights	local_manager	FILE:read
EACL entry	*token type*	*def.authority*	*value*
#2	access_id_GROUP	kerberos.V5	admin@ORG.EDU
	pos_access_rights	local_manager	FILE:read,write

Figura 2.6 – Exemplo de EACL na implementação de tokens

A verificação da autorização para *EACL* é feita através da *Generic Authorization and Access API* (*GAA-API*) que oferece um conjunto de funções com interface padronizada. Tal *API* deve ser ligada (*linked*) junto ao mecanismo de controle de acesso para ser efetivada [29]. Mais informações sobre EACL e GAA-API podem ser obtidas no Anexo VII.

Esta última abordagem − tanto para a representação como para a verificação de políticas de autorização − oferece recursos potencialmente mais poderosos e mais diretamente aplicáveis em sistemas de segurança que a anterior (*GACL*).

Como a verificação da autorização é feita por mecanismos que resguardam o objeto protegido, tomando como base atributos de autorização, em sistemas distribuídos tal atividade é normalmente executada no destino da solicitação de acesso, independente de

sua origem [5] [22] [29] [30].

Como já comentado anteriormente, a atividade de *evaluation* compreende o confronto dos atributos do principal com os *requirements* do objeto, sendo que o mecanismo de autorização, devido à natureza distribuída do sistema, precisa ter prova da autenticidade da solicitação de acesso. Assim, a verificação da autorização depende da comprovação da autenticidade do solicitante e da mensagem de requisição do acesso − o mecanismo de controle de acesso precisa estar integrado a um mecanismo de autenticação (seção 2.6.1). Neste caso, o serviço de autenticação, num sistema distribuído clássico, pode emitir além das informações de identificação sobre o principal, os atributos de autorização do mesmo em seus *certificados*.

Como comentado acima, devido à necessidade de comprovação da autenticidade nas trocas de mensagens e a natureza particular como cada máquina e domínio especifica sua política de autorização, os serviços de autenticação e de autorização podem ser combinados em sistemas distribuídos em diferentes configurações, visando o melhor tratamento da escalabilidade. Estas combinações presentes na literatura serão abordadas nos estudos de casos da seção 2.7.

2.7 Estudo de casos: Autenticação e autorização em sistemas distribuídos

Como já foi mencionado na seção 2.5, o conceito abstrato de monitor de referência é implementado por um núcleo de segurança. O conjunto de mecanismos de segurança − o núcleo de segurança, softwares, hardwares, controles criptográficos e outros controles adicionais que implementam as políticas de segurança de um sistema − são normalmente denominado **base computacional confiável** (*TCB –Trust Computing Base*) [31].

Neste texto esta-se limitando a caracterização do *TCB* ao conjunto de controles necessários para implementar a autorização e a autenticação. Assim, devido à complexidade (necessidade de diferentes *TCB*s), à natureza heterogênea e à dependência natural que há entre os serviços de autorização e de autenticação nos sistemas distribuídos, algumas configurações da disposição combinada destes serviços serão apresentadas a seguir.

2.7.1 Controle centralizado da autenticação e da autorização

As políticas de autorização de um sistema distribuído, através de seus serviços de autorização e autenticação, podem ser implementadas de forma centralizada. O mecanismo de controle centralizado é caracterizado por uma única entidade responsável pela autenticação e autorização (um *TCB*) para todo o sistema distribuído; a Figura 2.7 apresenta a seqüência de trocas envolvidas.

Inicialmente o principal deve cadastrar-se no sistema e, então, uma conta (*account*) é criada (este procedimento só precisa ser executado uma vez – linha 0, Figura 2.7). Posteriormente, o principal se autentica no serviço de autenticação (mensagem 1, Figura 2.7).

Quando o usuário pleitear um acesso (mensagem 3, Figura 2.7), o serviço de controle de acesso obtém as informações sobre o usuário a partir do serviço de autenticação e decide se permite ou nega o acesso; após esta decisão o usuário é informado que o acesso foi negado ou então o pedido é atendido (mensagem 4, Figura 2.7).

Nesta configuração a administração da política de segurança é uma atividade relativamente fácil, pois todos os controles estão centralizados em um TCB. Porém, os serviços de autorização e de autenticação podem apresentar um baixo desempenho provocado por esta centralização.

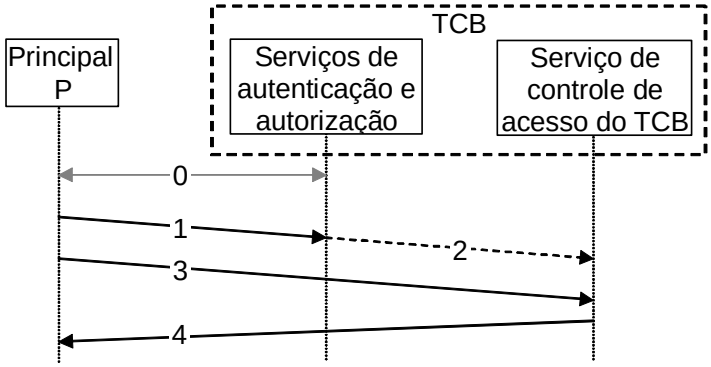


Figura 2.7 – Controles de segurança centralizados num sistema distribuído

Além disto, a solução não é facilmente escalável e muito possivelmente impraticável, considerando as dimensões que um sistema distribuído pode assumir. Estes controles tornam o *TCB* o ponto potencialmente deficitário em desempenho e/ou o ponto único de falhas de todo sistema distribuído. Conseqüentemente, a unicidade de tal controle tende a

transformá-lo no ponto vulnerável do sistema porque se atacado com sucesso compromete a segurança de todo o sistema distribuído.

2.7.2 Controle centralizado da autenticação e descentralização da autorização

Uma tendência clássica na concepção do controle de segurança em sistemas distribuídos é a centralização da autenticação, devido ao seu vínculo com o espaço de nomes e a descentralização da autorização através da distribuição das tarefas de administração de direitos de acesso e resguardo dos objetos – executadas localmente.

A escalabilidade é conseguida usando o conceito de domínio: os vários domínios em sistemas de larga escala devem impor suas políticas de autenticação e de autorização aos seus respectivos membros.

Nesta seção são abordadas três experiências que seguem esta filosofia na distribuição de suas funções de segurança[6]*: Kerberos Network Authentication Service, Framework de Autenticação X.509* e *Sistema Delta-4.*

A. Kerberos Network Authentication Service

O *Kerberos*, padrão *IETF* [20], segue o modelo de autenticação centralizada e autorização descentralizada. Neste caso, um servidor é utilizado para guardar as informações necessárias a intermediação do processo de autenticação no espaço de nomes do mesmo (domínio *kerberos*).

Um cliente/principal registrado no servidor *kerberos*, após trocar mensagens bem sucedidas com o mesmo, terá uma prova de sua autenticidade para apresentar aos servidores de aplicação que implementam os mecanismos de autorização e controle de acesso.

Principais e servidores de aplicação devem compartilhar a confiança na mesma entidade de autenticação (o servidor kerberos) para a efetivação dos acessos pleiteados aos objetos de aplicação do domínio. A decisão de permitir ou negar acessos será baseado na política de autorização que os servidores de aplicação implementam.

---

[6] O assunto controles criptográficos encontra-se no anexo I, para leitores não familiarizados com o mesmo.

A relação de confiança com o servidor *Kerberos* é definida pela posse de uma chave secreta: um principal pertencente a um domínio *kerberos* compartilha com o servidor *Kerberos* uma chave secreta (definida em tempo de criação da conta – *account* – do cliente no Kerberos – linha 0, Figura 2.8), cuja posse se constituirá em prova de autenticidade.

Em qualquer troca de mensagens com o principal, o servidor cifra mensagens enviadas com a respectiva chave secreta que ambos compartilham. Assim, por exemplo, quando um principal *P* desejar se comunicar com um servidor/principal *S*, este deve inicialmente se identificar junto ao servidor *kerberos AS* (processo de *login*: mensagem 1 na Figura 2.8).

Se *P* for bem sucedido no processo de *login*, terá um *token* (mensagem 2, Figura 2.8) permitindo-lhe se apresentar junto a um outro serviço *Kerberos* (o *TGS*), informando que deseja se comunicar com um servidor *S* (mensagem 3, Figura 2.8).

Após a verificação bem sucedida das credenciais de *P* o servidor *kerberos* (*TGS*) envia a este um conjunto de informações que servirão para certificação mutua entre *S* e *P* (mensagem 4 na Figura 2.8); dentre estas informações há um *ticket* que é um certificado de identificação e informações de sessão. Todas estas informações são cifradas com a chave secreta que o servidor *kerberos* compartilha com *P*.

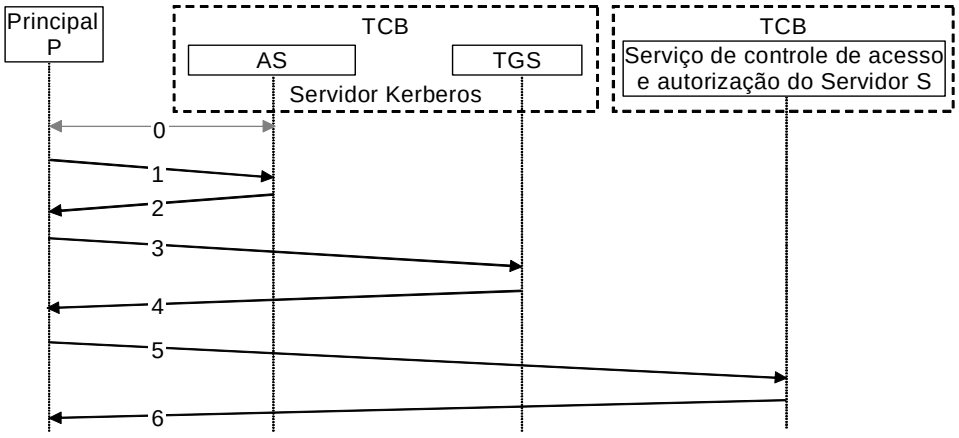


Figura 2.8 – Autenticação e autorização baseados no Servidor Kerberos

O principal *P* retem algumas destas informações, enviando o *ticket* e a requisição para *S*. O servidor *S* fará a verificação da autenticação, quando receber o *ticket* enviado pelo principal *P* (linha 5 na Figura 2.8).

Se o usuário tiver autorização suficiente e a verificação de autenticação ocorrer com sucesso, o acesso é permitido, em caso contrário, o principal é informado do fato

(mensagem 6, Figura 2.8). O *ticket* também pode transportar informações de autorização específicas para uma aplicação se assim for especificado no servidor *kerberos* (*TGS*). Maiores detalhes sobre as trocas envolvidas no protocolo implementado pelo servidor *kerberos* podem ser encontradas no Anexo II.

A escalabilidade no *kerberos* é conseguida usando domínios *kerberos* (*realms*). São previstas relações de confiança entre os servidores *Kerberos* de *realms* diferentes a partir do compartilhamento de chaves secretas entre cada par de servidores. Estas relações são tais que um principal de um domínio-*kerberos* pode ter acesso a serviços num outro domínio-*kerberos* desde que, de antemão, esteja estabelecida uma relação de confiança entre os servidores dos dois domínios.

O *kerberos* adota um sistema de nomes que concatena os nomes locais ao nome do domínio de modo a caracterizar a unicidade da representação dos identificadores (nomes) em operações inter-domínios.

De maneira geral esta proposta de descentralização da autorização no controle de segurança é bem aceita e utilizada em sistemas distribuídos. Porém, o servidor *kerberos* que trabalha com o sistema de chave secreta depende do compartilhamento de uma chave com cada principal pertencente ao domínio-*kerberos*. Além disto, por centralizar a autenticação, o *kerberos* pode se tornar o “gargalo” e/ou ponto único de falhas, ou ainda, o ponto potencialmente vulnerável do sistema.

B. Framework de Autenticação X.509

O modelo X.509 [21] é um *framework* que adota a hierarquia de nomes *X.500*, dando disponibilidade geográfica global aos objetos dos diversos domínios de nomes definidos a partir desta hierarquia.

A adoção da infra-estrutura de chave pública produz como resultado um sistema de autenticação centralizado onde o “servidor de autenticação” não intermedia o processo de autenticação, mas participa do mesmo quando necessário – produzindo certificados que associam uma chave pública a um determinado principal do domínio.

O *framework* de autenticação *X.509*, sendo baseado numa infra-estrutura de chave pública (Anexos I e III), assume que o principal possui um par de chaves – uma privada, que está em seu poder e outra pública. A chave pública foi armazenada num repositório para certificados no serviço de diretório *X.500* pela autoridade certificadora (*CA* -

Certification Autority) do domínio. Além da chave pública, os demais dados referentes à identificação e autorização do principal estão armazenados nestes repositórios em dois certificados: o de identificação e o de autorização, respectivamente, segundo a recomendação *X.509*-v3.

Assim, a recomendação *X.509* implementa o serviço de autenticação baseado num serviço de diretório *X.500* através da infra-estrutura de chave pública (*PKI* – Anexo III). Neste *framework,* o principal é identificado globalmente devido à hierarquia de nomes imposta pela recomendação *X.500* e pode assinar mensagens digitalmente com sua chave privada. Quando tais mensagens chegam ao destinatário (verificador), a assinatura digital deve ser validada através da chave pública do emissor, caracterizando assim sua autenticidade.

Para ilustrar o funcionamento do *X.509* considere o exemplo da Figura 2.9. Inicialmente, a *CA* recebe um pedido de geração do certificado de um principal *P* (mensagem 0, Figura 2.9), uma vez gerado, o mesmo é depositado no repositório do serviço de diretório *X.500* e devolvido ao principal (mensagem 1, Figura 2.9). A partir de então, durante a validade do certificado, o principal *P* pode enviar solicitações de acesso ao servidor *S*, as quais assina com sua chave privada (mensagem 2 na Figura 2.9).

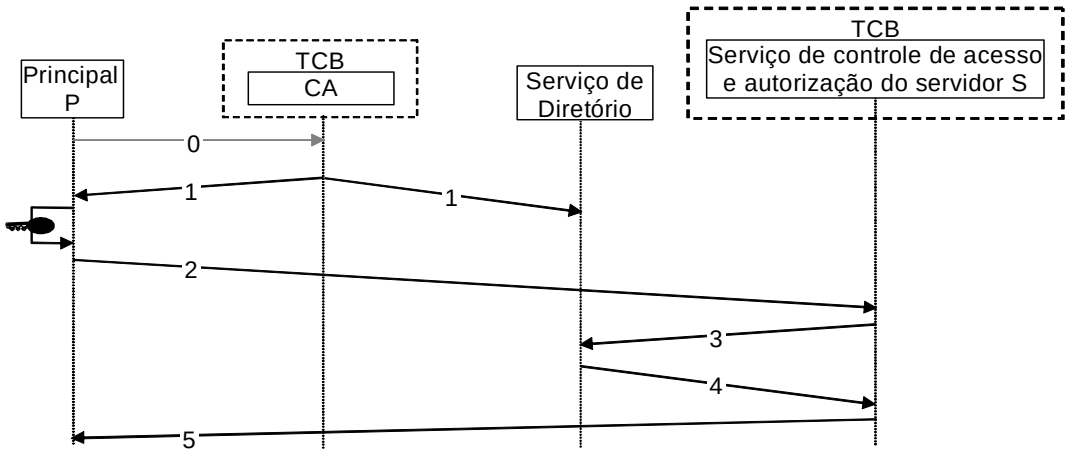


Figura 2.9 – Framework de Autenticação X.509

O servidor *S* ao receber uma solicitação de *P*, consulta o serviço de diretório (mensagem 3 na Figura 2.9) para obter a última *CRL* (*Certificate Revocation List* – Anexo III) e o certificado do principal *P*, se ainda não o possui. Então, de posse de ambos, verifica se o certificado ainda é válido e utiliza a chave pública de *P* para verificar a autenticidade

da solicitação. Com base nos atributos de autorização de *P* (mensagem 4, Figura 2.9 – disponíveis no certificado de autorização) e/ou na política de autorização, *S* atende ou nega a solicitação de acesso(mensagem 5, Figura 2.9). Maiores detalhes sobre as trocas envolvidas no *framework* podem ser obtidos no Anexo IV.

A adoção do serviço de diretório *X.500* que dispõe as *CA*s organizadas hierarquicamente dá âmbito global aos nomes *X.509*. Além disto, no *framework* são adotados os padrões *PKCS* (*Public Key Cryptosystem*) [32] e *PKI* (*Public Key Infrastructure* [33] que devido a sua ampla aceitação facilitam a interoperabilidade.

A grande vantagem desta abordagem é que cada principal tem em seu poder uma "única" informação que é guardada secretamente: a sua chave privada. Todas as outras informações são públicas.

A *CA* e o repositório de certificados (diretório *X.500*) não necessariamente participam intermediando o processo de autenticação – com o mesmo significado do servidor de autenticação do *Kerberos*. É evidente que no processo de verificação um servidor de aplicação pode ter acesso ao serviço de diretório na busca de certificados de principais, mas uma vez de posse destes certificados o servidor pode guardá-los para uso em acessos posteriores do mesmo principal.

O servidor de aplicação só precisa fazer consultas freqüentes a *CRL*, que também pode prever alternativas na sua publicação, evitando o gargalo de pontos de acesso únicos no domínio. Deste modo, um sistema baseado em *X.509* prevê alternativas para o processo de autenticação visando diminuir a dependência de uma entidade única centralizadora no domínio.

Apesar destes esforços para minimizar esta dependência de uma entidade única no armazenamento e na dinâmica da autenticação e, mesmo a *CA* operando *offline* no sentido de se tornar menos vulnerável, o risco de vulnerabilidade e falha não é afastado porque qualquer operação incorreta, inadequada ou não autorizada na *CA* continuará podendo comprometer todo o sistema.

C. Sistema Delta-4

Os mecanismos kerberos e X.509 são amplamente difundidos e utilizados nos dias atuais. Porém, outras propostas na construção do controle centralizado da autenticação associado com a distribuição da autorização existem na literatura. A base de segurança

desenvolvida no sistema *Delta-4* [34] é exemplo destas propostas.

O sistema *Delta-4* foi projetado tendo premissas de tolerância a faltas e a intrusões. Estas premissas no sistema *Delta-4* (Figura 2.10) estão fundamentadas no princípio da fragmentação e disseminação [35], onde a fragmentação é utilizada para particionar a informação sensível e a disseminação é usada no armazenamento dos respectivos fragmentos em locais distribuídos geograficamente.

A fragmentação e a disseminação são conduzidas de tal maneira que a invasão de certo número de sítios de armazenamento destas informações sensíveis não implica na violação das propriedades de segurança das mesmas.

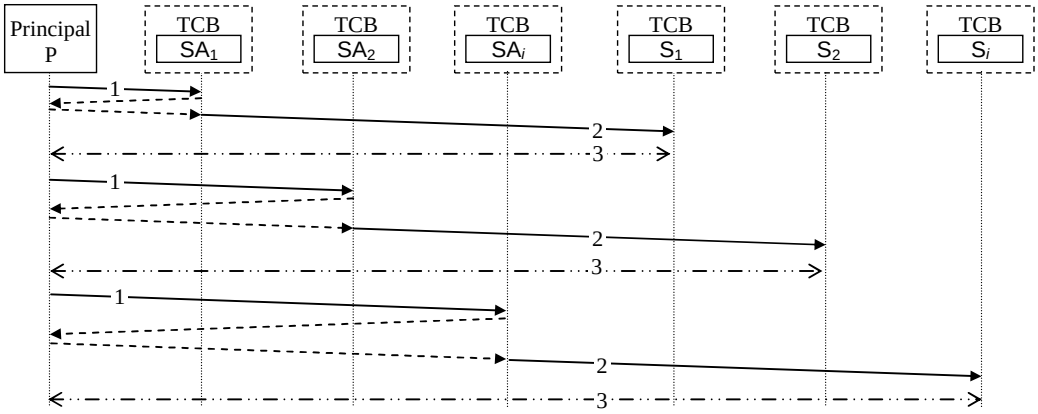


Figura 2.10 – Tolerância a Intrusões e a Faltas no Sistema Delta-4

Assim, chaves e outras informações de acesso são fragmentadas e disseminadas entre servidores de autenticação (*SA*s). Informações das aplicações sofrem o mesmo processo de fragmentação e disseminação em um conjunto de servidores de aplicação (servidores S_i, i = 1,2,3, ..., Figura 2.10).

Na recuperação das informações das aplicações armazenadas no grupo de servidores S_i, o principal precisa inicialmente se autenticar com sucesso num número mínimo de serviços de autenticação para poder reconstituir as chaves de acesso (mensagem 1 da Figura 2.10). Em seguida, de posse das chaves recuperadas, o principal pode ter acesso aos fragmentos da informação de aplicação, passando pela autorização distribuída dos servidores S_i (mensagem 2, da Figura 2.10).

A informação original é reconstituída a partir dos fragmentos, também com o uso das chaves recuperadas (mensagem 3, Figura 2.10). Detalhes dos algoritmos de fragmentação e disseminação podem ser conseguidos em [35].

Este sistema adiciona características que melhoram a confidencialidade da informação, porque o intruso precisa invadir um número mínimo de sítios para conseguir recuperar os fragmentos da informação e reconstruí-la. Além disto, a disponibilidade da informação também é melhorada porque o mesmo fragmento é replicado e disseminado em um grupo de máquinas. Todas as réplicas de um mesmo fragmento teriam que ser alteradas para comprometer a integridade da informação do mesmo. A replicação também permite minimizar a negação de serviço, porque se um usuário não tem acesso a um fragmento em um sítio poderá tê-lo em outro onde se encontra uma réplica do mesmo.

Em suma, as proposições no sistema *Delta-4* são uma infra-estrutura dotada de serviços de autenticação e autorização com as tolerâncias a faltas e a intrusões integradas para garantir o serviço de segurança do sistema como um todo. Esta proposta mantém as dificuldades de autenticação centralizada no que se refere a característica de solução não escalável.

2.7.3 Controle descentralizado da autenticação e da autorização

As configurações mostradas anteriormente apresentam o inconveniente da centralização de algumas atividades. Assim, o ideal parece ser a descentralização de ambas as atividades de controle: a autenticação e a autorização. A maneira mais natural para integrar estas funções com implementações distribuídas é através do uso de *redes de confiança*. Ou seja, teríamos várias entidades confiáveis distribuídas no sistema que se encarregariam da implementação de políticas de autenticação e autorização.

Duas abordagens que se baseiam em redes de confiança para integrar a autenticação e autorização em sistemas distribuídos são as propostas ***TCSEC*** (*Trusted Computer System Evaluation Criteria*) e ***SPKI*** (*Simple Public Key Infrastructure*). O conceito de rede de confiança no contexto do *SPKI* será retomado no capítulo 3.

Esta seção discutirá os aspectos relacionados ao modelo *TCSEC* [31] – uma proposta de descentralização do controle de autenticação e autorização. No *TCSEC* os serviços de autenticação e de autorização têm seus controles distribuídos, dispostos sobre o conjunto de *TCBs* que compõem a rede de confiança da base computacional (*Network Trust Computing Base – NTCB*).

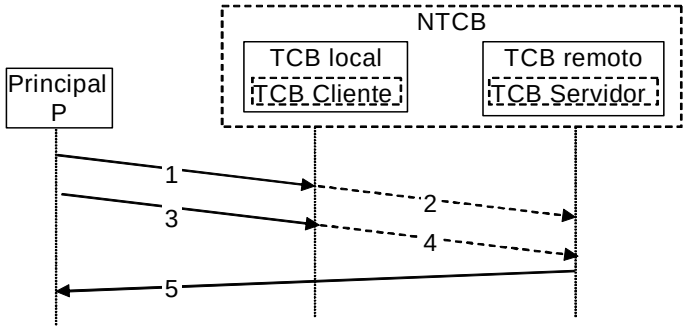


Figura 2.11 - Proposta TCSEC de descentralização do controle de segurança

Cada máquina do sistema tem o seu *TCB* que intermedia os acessos dos sujeitos locais à objetos locais ou remotos. Cada *TCB* realiza as funções de autenticação de seus sujeitos.

Os acessos remotos envolvem trocas entre *TCBs* através do *NTCB*. Supondo que um principal no domínio de um *TCB* (*TCB* cliente) deseja ter acesso a um objeto de um outro *TCB* (*TCB* servidor), neste caso, basta o principal se autenticar (mensagem 1, Figura 2.11) e fazer a solicitação (mensagem 3 da Figura 2.11) ao *TCB* local (*TCB* cliente). O *TCB* local encaminhará a solicitação ao *TCB* remoto. Tal operação é possível porque há uma rede de confiança através da *NTCB*, formada pelos *TCB*s, implementado a autenticação e um mecanismo de autorização "global" controlando todos os objetos disponíveis na *NTCB*.

Neste modelo o particionamento da estrutura de autorização e autenticação pode comprometer a segurança do sistema como um todo, porque o modelo é baseado na confiança entre *TCB*s*;* estes componentes isolam e intermediam as interações entre sujeitos e objetos servidores de aplicação. Basta que apenas uma partição da matriz de acesso seja corrompida – um *TCB* – para que a segurança de todo o sistema fique comprometida. No modelo *TCSEC* não são descritos procedimentos, nas interações entre *TCB*s, que permitam a detecção de elementos corrompidos na rede *NTCB*.

2.7.4 Comparação entre as configurações de controle da autenticação e autorização

De maneira geral, já foram observadas as características e limitações relevantes das configurações de controle da autenticação e da autorização em sistemas distribuídos nas seções de 2.7.1 a 2.7.3. A Tabela 2.3 resume os aspectos relevantes dos mecanismos de autenticação e autorização em sistemas distribuídos.

Abordagem	Autenticação	Autorização	Espaço de nomes	Escala-bilidade	Interope-rabilidade	Tolerância a faltas
Kerberos	centralizada	descentralizada	local	sim	sim	--
X.509	centralizada	descentralizada	global	limitada	sim	--
Delta-4	centralizada	descentralizada	local	limitada	sim	sim
TCSEC	descentralizada	descentralizada	local	limitada	---	--

Tabela 2.3 – Resumo das características dos modelos de controle de segurança

O modelo de autenticação centralizada e autorização descentralizada (seção 2.7.2) apresenta o problema da centralização de qualquer função em sistemas distribuídos: ponto único de falha, vulnerabilidade e/ou comprometimento do desempenho (gargalo) do sistema nas funções centralizadas. Evidentemente, tal tipo de configuração cria uma dependência da entidade central que a torna o ponto potencial onde "algo não esperado pode ocorrer" e todo o esquema de segurança do sistema pode ficar comprometido.

Como visto nas abordagens que seguem este modelo (*kerberos*, *X.509* e *Delta-4*), algum tipo de mecanismo ou técnica é utilizado no sentido de minimizar os efeitos da centralização. No caso, o servidor *kerberos* pode ser considerado como o mais limitado devido às restrições impostas pelo uso do sistema de chave secreta. No *X.509*, baseado no sistema de chave pública e no serviço de diretório para publicação de certificados, o mecanismo de autenticação pode ter minimizado o problema de centralização. É possível, por exemplo, a replicação do diretório, mas a representação global de nomes (*X.500*) sofre normalmente críticas pela sua complexidade.

Seguramente, as técnicas de tolerância à faltas e a intrusão adotada no sistema *Delta-4* é que verdadeiramente implementam alternativas para minimizar os efeitos da centralização, porém a dinâmica de funcionamento do sistema como um todo parece criar certa complexidade ao seu uso em sistemas de larga escala.

A configuração de autenticação e autorização aplicando o modelo descentralizado mostra-se mais adequada para a integração dos dois serviços (autenticação e autorização) em sistemas distribuídos de larga escala, porque está fundamentado na idéia de não criar um ponto central potencialmente deficiente em aspectos como os comentados no parágrafo anterior. Porém, na proposta de descentralização do *TCSEC* não são aprofundados os aspectos das interações na rede de confiança, o que limita talvez a sua aceitação como modelo.

2.8 Considerações finais

Neste capítulo foram apresentados conceitos que fundamentam a segurança em sistemas computacionais. O texto deste capítulo se fixou principalmente nos tipos de políticas e modelos de segurança, citando suas características e limitações. Foram vistas, também, as principais abordagens clássicas para a autenticação e a autorização em sistemas distribuídos.

No próximo capítulo será explorado melhor o conceito de redes de confiança como base para a implementação de serviços de autenticação e autorização em sistemas distribuídos de larga escala. Modelos fundamentados neste conceito devem preencher requisitos como: a gestão distribuída das políticas de segurança, interoperabilidade e tolerância a falhas – aspectos importantes dos sistemas distribuídos de larga escala.

3. Autenticação e autorização orientadas a chaves

3.1 Introdução

No capítulo anterior foram expostas as abordagens normalmente usadas no controle de autenticação e autorização, enfatizando as suas características de disposição em sistemas distribuídos. As mais usadas trazem sempre algum tipo de prejuízo devido a centralização causada pela dependência de um espaço de nomes que impõem dificuldades à escalabilidade.

Neste capítulo, é mostrada uma alternativa para estes serviços que é baseada na noção de cadeias de confiança, construídas sobre a infra-estrutura de chaves públicas, permitindo descentralizar por completo os controles de autenticação e de autorização.

3.2 Gerência de confiança

Os modelos e abordagens de autenticação e autorização em sistemas distribuídos, descritos no capítulo anterior, na implementação dos serviços de autenticação dependem de domínios de nomes, o que implica na centralização destes controles em cada domínio específico.

O modelo centralizando controles por domínio é adequado em redes corporativas, mas apresenta certos problemas quando o ambiente é a rede mundial, onde o cliente (usuário) muitas vezes não é conhecido de antemão. O modelo de confiança (*trust model*)

baseado em uma entidade centralizadora de autenticação impõe sérias restrições a escalabilidade e a flexibilidade em sistemas distribuídos.

Abordagens que se propõem a atender os requisitos de escalabilidade e flexibilidade estão baseadas em infra-estruturas de chaves públicas ou *PKIs* (*Public Key Infrastructures*). Atualmente, a mais comum destas infra-estruturas é o *X.509*. O modelo de confiança do *X.509* é hierárquico. Comunidades *X.509* devem ser construídas com base na confiança das chaves privadas das *CA*s, organizadas em uma hierarquia global. Uma *CA* controla sua chave privada, usada na assinatura dos certificados emitidos pela mesma. Estes certificados vinculam nomes de validade global às respectivas chaves públicas. O modelo, essencialmente, baseia-se em certificados formando cadeias (*chain*s) de autenticação a partir da chave de uma *CA* confiável até uma chave pública de um usuário (principal).

Quando são consideradas aplicações na Internet, a autorização e a autenticação devem evoluir tomando como base modelos onde às relações de confiança possam ser estabelecidas de maneira distribuída – favorecendo as soluções escaláveis e flexíveis no que se refere à segurança.

Na infra-estrutura *X.509*, é possível construir relações de confiança através de certificação cruzada entre duas *CA*s, objetivando dispensar a necessidade de percorrer toda a estrutura hierárquica para ir de uma *CA* até outra [36]. Porém, em tal modelo, este tipo de alternativa de fluxo só é permitido entre autoridades certificadoras, o que evidentemente não descaracteriza a *CA* como entidade centralizadora da certificação nem a rigidez e complexidade da estrutura hierárquica imposta por esta infra-estrutura.

Para o cenário de aplicações como os da rede mundial, seria mais adequado um modelo onde às relações de confiança pudessem ser estabelecidas através de cadeias de confiança, sem nenhuma entidade centralizadora de certificação, como se cada entidade da cadeia (*chain*) fosse uma *CA* do modelo de *PKI*.

Na infra-estrutura *PGP* (*Pretty Good Privacy*) [37], desenvolvida por *Phil Zimmermann* em 1991, para cifragem e autenticação de arquivos e correio eletrônico é utilizada criptografia de chave pública. No *PGP* a estrutura para gerenciamento e certificação de chaves públicas está baseada na chamada teia de confiança (*web of trust*). A **teia de confiança** não se baseia em *CA*s centralizadas como o *X.509*; com a infra-estrutura do *PGP* os seus usuários podem construir caminhos de confiança de forma arbitrária

através da comunidade *PGP* ao redor do mundo. Um caminho de confiança liga um principal a outro principal através da teia de confiança.

Um certificado de nome no *PGP* também liga um nome global a uma chave pública. Qualquer chave pode emitir um certificado; os certificados não são emitidos por *CA*s com responsabilidades legais, mas por usuários comuns. Na teia de confiança, vários usuários assinam cada certificado. O modelo de teia de confiança funciona bem para pequenas comunidades ou comunidades que interagem de maneira pouco freqüente, mas sofre do problema de requerer múltiplas assinaturas para tomar uma decisão de confiança. Além disso, diferentes chaves privadas podem ser controladas por um usuário [38], assim como em *X.509*.

O controle de acesso é tradicionalmente baseado na identidade autenticada do cliente que deseja acessar um recurso e em uma *ACL* (lista de controle de acesso) armazenada na memória local de um *guardião* (núcleo de segurança) do recurso acessado.

Para a Internet, quando usadas às infra-estruturas citadas acima, a autenticidade da identidade dos clientes baseia-se em chaves públicas e certificados de nome. Certificados *X.509* ou *PGP* ligam nomes a chaves públicas. Estas infra-estruturas e esquemas de autorização montados a partir destes certificados de nomes são classificados como **orientados a nomes.** Blaze e seus colegas [38] argumentam que o uso de infra-estruturas de chaves públicas baseadas em nomes juntamente com *ACL*s são uma solução inadequada para sistemas como a rede mundial. As dificuldades com a escalabilidade e a falta de flexibilidade, nestas infra-estruturas, são o destaque quando tratamos com nomes globais. Além disto, a necessidade da hierarquia de *CA*s agindo como entidades legais em diferentes países é apontada como uma das principais dificuldades do *X.509*.

Como alternativa em [38] é introduzido o conceito de **gerência de confiança** (*trust management*) como um conjunto de mecanismos unificados para especificar e validar políticas de autorização (*ACL*s), credenciais e seus relacionamentos. Estes mecanismos de gerência de confiança são muitas vezes chamados de **orientadas a chaves**. Ou seja, as credenciais em abordagens de gerência de confiança ligam diretamente chaves a autorizações.

Assim, a gerência de confiança unifica a noção de política de segurança, credenciais, controle de acesso e autorização. Além disto, as políticas e credenciais são escritas numa linguagem ou em estruturas que são compartilhadas por todas as aplicações nos sistemas se

utilizando deste conceito.

Na literatura são distinguidas duas abordagens para a gerencia de confiança. Em uma abordagem, a gerência de confiança é estabelecida usando uma linguagem que pode ser usada na descrição da política e credenciais; faz parte ainda desta gerência de confiança um motor-lógico que define o comportamento do módulo de verificação de conformidade (*compliance checker*). Na outra abordagem, uma estrutura de dados é usada de forma padronizada para descrever os atributos de segurança no sistema, definindo a política e os certificados do mesmo. Em ambas as abordagens, a verificação de conformidade indica se os certificados (credenciais) apresentados por um principal estão de acordo com a política especificada localmente.

Exemplos de sistemas baseados em gerência de confiança incluem o *PolicyMaker* [38] [39], o *KeyNote* [40] e o *Simple Distributed Security Infrastructure / Simple Public Key Infrastructure* (*SDSI/SPKI*) [41] [42].

3.2.1 PolicyMaker

No ambiente do *PolicyMaker* o objetivo é a autorização, expressa através de asserções. Uma asserção é um par (*f, s*), onde *s* representa a autoridade origem (emissor) e *f* descreve a autorização concedida e o seu beneficiário. A política e as credenciais expressas na forma de asserções representam basicamente a mesma estrutura, diferindo apenas quanto ao emissor. Para a política, *s* é sempre a palavra *POLICY* e para a credencial, o emissor *s* é uma chave pública. Além disto, as asserções de política não são assinadas porque a política é local ao módulo verificador de conformidade; as credenciais precisam ser assinadas pelo emissor e conseqüentemente verificadas pelos destinatários antes de serem efetivamente utilizadas.

Na especificação das asserções, uma linguagem ou especificação formal deve ser usada de modo a fornecer um meio de expressão e de interpretação correta e inequívoca. Esta ferramenta é necessária porque um ambiente local, nas decisões de verificação, pode importar credenciais de diversas entidades. O *PolicyMaker* não define uma linguagem única para a especificação de asserções; sua principal preocupação é com o formalismo da especificação.

A verificação de conformidade é independente das linguagens de especificação de

políticas e credenciais. O formalismo usado permite também que o motor-lógico do módulo de verificação de conformidade seja o mínimo possível, para facilitar sua inspeção. Além disto, boa parte das atividades envolvidas na gerência de confiança, como a verificação de assinaturas, é atribuída à aplicação que invoca tal módulo, permitindo a esta a livre escolha da tecnologia de segurança a ser utilizada.

Basicamente, o módulo de gerência de confiança (módulo de verificação de conformidade) toma como entrada um conjunto de asserções de credenciais (C), requisições (r) e asserções de política (P) e deverá gerar como saída um parecer refletindo a conformidade ou não de *C* e *r* com *P*.

As credenciais em *C* devem formar cadeias de delegações que dão ao requerente as permissões para o acesso desejado. Não é função do módulo descobrir as credenciais que faltam em *C* e buscá-las; este apenas decide quais e em qual ordem as asserções devem ser avaliadas para gerar todos os registros parciais de aceitação. Para isto o módulo usa uma estratégia de buscas repetidas que geram os registros parciais. No fim do processo tais registros são ordenados para que o módulo possa concluir se os mesmos representam um conjunto substancial de provas de conformidade para a requisição em questão ou não.

Em [39] é mostrado com detalhes a especificação formal do problema de verificação de conformidade. Em tal texto, é mostrado que o problema geral de verificação apresenta uma complexidade computacional sem solução polinomial (*undecidable*). Ao mesmo tempo, é mostrado que em casos especiais o problema possui solução em tempo polinomial; estes casos especiais podem representar uma vasta variedade de aplicações.

O *PolicyMaker* tem sua principal limitação no problema de verificação de conformidade, onde a garantia de sua correção formal para a versão computacionalmente viável só pode ser alcançada se todas as asserções forem monotônicas[7] (*monotonic*). Isto exclui a possibilidade de uso de "credenciais negativas" como as *CRL*s[8], por exemplo. Porém, em geral as asserções de políticas negativas podem ser substituídas por asserções, por exemplo, que podem exigir do interessado a apresentação de prova de não inclusão em *CRL*s. De maneira geral, a utilização de políticas com especificações monotônicas evita as

[7] Asserções monotônicas associam uma ação a um conjunto de evidências que uma vez provadas, automaticamente, aprovam tal ação num superconjunto dessas evidências.

[8] CRL – *Certificate Revocation Lists* – Lista de revogação de certificados.

situações de especificação com regras conflitantes.

3.2.2 KeyNote

O *KeyNote* foi desenvolvido sob os mesmos princípios do *PolicyMaker*. Porém, uma linguagem específica para definição de asserções foi adotada por questão de eficiência e para facilitar a interoperabilidade na escrita de políticas e credenciais. Isto permitiu passar mais atribuições para o módulo de gerência de confiança; a verificação de assinatura digital, por exemplo.

Na busca da prova de conformidade foi adotado o algoritmo *DFS* (*Depth First Search*) que é executado no conjunto de credenciais apresentadas pelo requerente – o encadeamento dessas credenciais permite que a estrutura resultante seja tratada como um grafo direcionado; diferente da estratégia de buscas repetidas do *PolicyMaker* que geram os registros parciais utilizados na construção da prova de conformidade.

Por ser um ambiente mais específico que o *PolicyMaker*, o *KeyNote* é mais aplicável, porém, apresenta as mesmas limitações que o anterior. Maiores detalhes sobre o *KeyNote* podem ser obtidos em [40]. No *KeyNote*, também, o problema da busca/descoberta da cadeia de certificados é atividade a ser executada em paralelo pelo requerente.

3.3 SPKI (Simple Public Key Infrastructure) / SDSI (Simple Distributed Security Infrastructure)

A infra-estrutura *SDSI/SPKI* foi motivada pela percepção da complexidade do *X.509* devido ao seu esquema global de nomeação. O *SPKI* [41] e o *SDSI* [43] são duas propostas com propósitos complementares: O *SDSI*, projetado no *MIT* por *Ronald Rivest* e *Butler Lampson*, é uma infra-estrutura de segurança com o objetivo principal de facilitar a construção de sistemas distribuídos seguros e escaláveis. *Carl Ellison*, por sua vez, conduziu esforços no projeto de um modelo de autorização simples, bem definido e implementável que tomou o nome de *SPKI*.

A combinação das duas propostas (SDSI e SPKI) forma uma base para a autenticação e autorização em aplicações distribuídas, citada por alguns autores apenas

como *SPKI*. Nesta infra-estrutura, simples e flexível, os espaços de nomes de principais[9] são locais e o modelo é baseado em cadeias de confiança. No contexto deste trabalho será considerada a versão 2.0 de *SDSI/SPKI* .

Em *SDSI/SPKI* há dois tipos distintos de certificados, um para nomes e outro para autorizações. Os certificados de nomes ligam nomes a chaves públicas ou, ainda, a outros nomes. O sistema de nomeação é adotado do *SDSI* e induz ao uso de nomes locais mesmo no sentido global de um ambiente distribuído. Ou seja, no lugar de criar um espaço de nomes global único, os nomes *SDSI/SPKI* são sempre locais, correspondendo ao espaço de nomes do emissor do certificado. O emissor do certificado é sempre identificado pela sua chave pública. A combinação chave pública mais nome local forma um identificador global único.

No *SDSI/SPKI* é usado um modelo igualitário: os principais são chaves públicas que podem assinar e divulgar certificados, como uma *CA* do *X.509*. Assim, qualquer principal pode criar seu par de chaves (privada e pública), associar à chave pública do par a um nome no seu espaço local de nomes e divulgá-la através de um certificado. Não há uma entidade centralizadora para registro de chaves públicas e emissão de certificados como a *CA* da *PKI X.509*. Assim, cada principal define da maneira que lhe parecer mais intuitiva, em seu espaço de nomes, os nomes que deseja atribuir aos outros principais de seu relacionamento. Só o principal que emite um certificado pode revogá-lo.

Um certificado de nome pode fazer referência a um nome publicado em outro certificado – pertencente ao espaço de nomes de outro principal e assim sucessivamente, de modo a formar uma cadeia de certificados de nome. Assim, a divulgação de nomes no *SDSI/SPKI* é feita através de cadeias de confiança formadas por certificados de nomes ligados por encadeamento de referências [44]. Estas cadeias de nomes devem ser reduzidas a uma chave pública que representa o principal sendo referenciado quando se deseja a identificação do mesmo. Em SDSI/SPKI, a divulgação de principais é feita pela propagação de certificados de nomes.

Considerando que um certificado de nomes no *SDSI/SPKI* pode associar nomes a

[9] São entidades ativas que possuem um par de chaves (privada e pública) e são capazes de executar assinaturas digitais. A chave pública neste caso é utilizada para verificação da assinatura digital e para identificar univocamente o principal no espaço global.

chaves públicas ou a outros nomes, o certificado de nomes, logicamente, pode ser expresso da seguinte forma:

K_x "*nome*" → $K_{\text{"}a\_definir\text{"}}$,

Neste caso, K_x "*nome*" indica que "nome" está definido no espaço local de nomes de K_x[10] e $K_{\text{"}a\_definir\text{"}}$ representa a chave pública sendo associada ao referido "nome". Assim, tomando *x* como *Joãozinho* (com chave pública K_J) e considerando que o mesmo deseja emitir um certificado de nomes para a sua mãe ($K_{Mãe}$ é a chave pública de sua mãe), então, a definição do nome "mãe" no espaço de nomes de Joãozinho será publicada no certificado como:

K_J "mãe" → $K_{Mãe}$.

Considerando agora que *Alice* (chave pública K_A) conhece a mãe de Joãozinho e deseja definir um nome (Maria) para se referir a mesma em seu próprio espaço de nomes. Então, de maneira análoga, Alice define o nome "Maria" em seu espaço de nomes se referindo ao certificado emitido por *Joãozinho*:

K_A "Maria" → K_J "mãe".

Percebe-se pelo exemplo acima que a divulgação de nomes em *SPKI* é feita através de cadeias de certificados que determinam a validade global destas identificações.

Os certificados de autorização *SDSI/SPKI* ligam autorizações (direitos de acesso) a chaves, nomes ou "grupos especiais" de principais (*threshold subjects*). O emissor de um certificado é um principal (sujeito comum, administrador ou guardião de um serviço) que cria um certificado para delegar permissões de acesso a outros principais no sistema.

O emissor de um certificado pode permitir que o sujeito (principal receptor) delegue as permissões recebidas a outros principais, através de outros certificados de autorização que devem ser anexados a cadeia. Ou seja, o sujeito pode no papel de emissor delegar direitos recebidos a outros principais. O modelo de confiança do *SDSI/SPKI* possibilita a construção de cadeias de autorização que partem da chave do guardião de um serviço e

[10] Principais são chaves públicas no *SPKI*, no caso, a notação sinaliza a chave pública *K* do principal *x* (K_x).

terminem nas chaves dos principais (clientes).

Evidentemente, para que um principal faça a delegação de permissões de acesso, este deve confiar o suficiente no principal (sujeito) ao qual está fazendo a delegação. Para isto, um certificado de autorização contém um bit de delegação (*true*/*false*) especificando se as permissões podem ser repassadas ou não. Assim, o sujeito (receptor/favorecido) pode repassar através de outro certificado a íntegra das permissões que lhe foram conferidas ou um subconjunto destas. Delegações sucessivas envolvendo as mesmas permissões ou um subconjunto dessas definem a propagação de autorização e a formação de uma cadeia de certificados de autorização que é representada na Figura 3.1 pelas linhas 0, 0’ e 0’’.

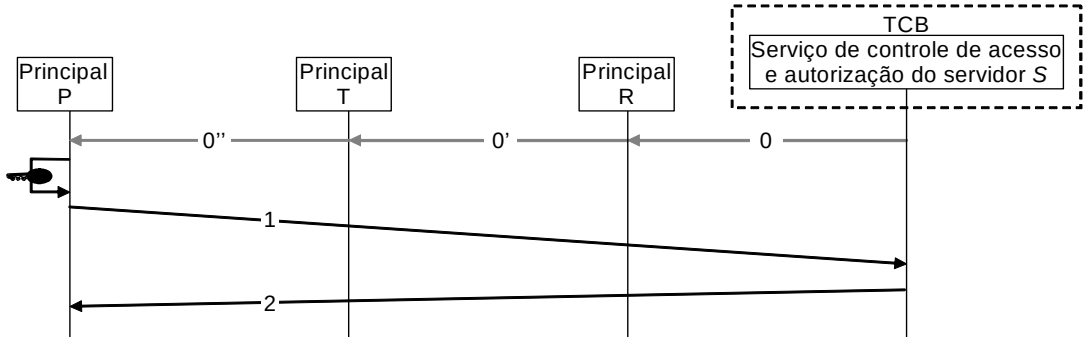


Figura 3.1 - Delegação de autorização e cadeia de confiança no SDSI/SPKI

Junto com o certificado concedendo as permissões ao sujeito, o emissor deverá fornece a seqüência de certificados antecedendo-o na delegação de tal autorização. Assim, ao receber um certificado com delegação permitida o sujeito poderá repassá-la a frente. Ao propagar tal concessão deverá ser anexada a seqüência de certificados recebida mais um certificado (o seu), e toda a cadeia deverá ser encaminhada ao sujeito. As concessões de autorização em *SDSI/SPKI* não são revogáveis e valem enquanto o período de validade do certificado estiver vigente.

A infra-estrutura *SDSI/SPKI* apresenta outras características. *ACLs* podem ser construídas com uma sintaxe similar a de um certificado de autorização, identificada pela diferença básica de não possuir um emissor do certificado, por serem locais ao guardião do serviço. Em linhas gerais *SDSI/SPKI*, no que se refere aos certificados e *ACLs*, define um formato único de representação facilitando as atribuições e verificações de autenticação e autorização.

3.3.1 Redução de cadeias de certificados

Como comentado anteriormente no *SDSI/SPKI* a divulgação do "nome" de um principal é feita através da emissão de certificados de nomes, sendo que os principais são identificados como chaves públicas. As cadeias de certificados de nomes, que resultam da publicação sucessiva de diferentes certificados de nomes referindo-se ao mesmo principal, devem ser reduzidas à chave pública representando o principal sendo referenciado.

De maneira similar a propagação de permissões, feitas por delegações sucessivas de direitos, forma também cadeias de certificados de autorização, que devem ser reduzidas a um certificado único para identificar o sujeito e suas permissões.

A seguir são mostradas as regras recomendadas pela *IETF* [42] para a redução de cadeias de nomes e de autorização.

A. Redução de cadeias de nomes

Um certificado de nome é composto por uma quádrupla (*4-tuple*) com os elementos indicados na Tabela 3.1.

Campos	**Descrição**
Emissor (*Issuer*)	chave pública da entidade (chave emissora) que está definindo o "Nome" no seu espaço local.
Nome (*Name*)	nome local que está sendo atribuído ao sujeito.
Sujeito (*Subject*)	uma chave pública ou a identificação de uma entidade definida em outro espaço de nomes que está sendo redefinida (referenciada) no espaço de nomes local ao emissor.
Data de validade (*Validity dates*)	especificação do período de validade do certificado, em formato 'data-hora'.

Tabela 3.1 – Certificado de nomes SDSI/SPKI

Considerando o exemplo da seção anterior, onde o nome "mãe" definido no espaço de nomes de *Joãozinho* (K_J) tinha uma chave atribuída através de um certificado de nomes representado de forma simplificada como: K_J *"mãe"* $\rightarrow K_{Mãe}$ e assumindo os campos da Tabela 3.1, o certificado de nomes correspondente toma a forma da quádrupla:

(K_j, "mãe", $K_{Mãe}$, "data-hora").

Considerando ainda o mesmo exemplo, onde *Alice* referenciava a mãe de *Joãozinho* com o nome local "Maria", uma segunda definição de nomes é feita no espaço de nomes locais (K_A *"Maria"* $\rightarrow K_J$ *"mãe"*). Esta definição, baseada na Tabela 3.1, assume a forma:

(K_A, "Maria", K_J *"mãe"*, "data-hora").

No exemplo exposto, o objetivo da redução da cadeia de certificados formados pelas quádruplas (K_A, "Maria", K_J *"mãe"*, "data-hora") e (K_j, "mãe", $K_{Mãe}$, "data-hora") é chegar ao certificado:

(K_A, "Maria", $K_{Mãe}$, "data-hora").

Em suma, o primeiro certificado associa um nome (mãe) a uma chave no espaço de K_J, o segundo certificado por sua vez associa um outro nome (Maria) ao primeiro K_J ("mãe"). Logo, estes dois certificados podem ser substituídos por um terceiro ligando diretamente o identificador K_A *"Maria"* à chave pública $K_{Mãe}$[11].

B. Redução de cadeias de certificados de autorização

O certificado de autorização é uma quíntupla (5-tuple) com os elementos indicados na Tabela 3.2.

Considerando que se deseja reduzir a cadeia formada pelos certificados (E1, S1, D1, A1, V1) e (E2, S2, D2, A2, V2), então, o sujeito de um certificado deve ser o emissor do outro (S1 = E2) e o bit de delegação do primeiro certificado deve permitir a delegação (D1 = *True*), com isto o processo de redução deve levar ao certificado:

[E1, S2, D2, *intersect* (A1,A2), *intersect* (V1,V2)]

Neste caso, *intersect* no campo de autorização representa a interseção (*string* a *string*) dos elementos de *A1* e de *A2*. Em relação ao campo validade é necessário que *intersect* (V1,V2) considere a interseção dos períodos de validade dos certificados em questão. Se não houver interseção a função de redução deve retornar uma exceção.

O *SDSI/SPKI* prevê ainda recursos para tolerância a intrusões e a faltas. São introduzidos apenas para os certificados de autorização, os *threshold subjects* cujo objetivo é garantir que no mínimo *k* dentre *n* sujeitos autorizem a delegação de permissões. Assim, se para acessar um objeto protegido dois dentre três principais precisam assinar uma requisição, um intruso conseguindo de alguma forma uma das chaves privadas dos três

[11] Na RFC 2693 [42] são sugeridas regras gerais de redução que podem ser usadas em casos específicos como o apresentado no texto acima. Estas regras são apresentadas abaixo onde K*i* representa uma chave pública e Nz um nome local: [(K1 N N1 N2 N3) + ((K1 N) → K2)] → (K2 N1 N2 N3) e [(K1 N Na Nb Nc) + ((K1 N) → (K2 N1 N2 ... Nk))] → (K2 N1 N2 ... Nk Na Nb Nc)

sujeitos certificadores da requisição não conseguirá ainda o acesso ao recurso.

Campos	Descrição
Emissor (*Issuer*)	chave pública da entidade que emitiu o certificado ou a palavra "*SELF*" se o certificado for uma *ACL* emitida pelo guardião.
Sujeito (*Subject*)	chave pública (ou *hash* dessa) ou nome identificando o principal que receberá a autorização.
Delegação (*Delegation*)	valor booleano (*True*/*False*), indicando se o sujeito pode ou não propagar a autorização que lhe foi delegada pelo emissor.
Autorização (*Authorization*)	contém as permissões concedidas pelo emissor, representadas como uma *S-expression*.
Validade (*Validity dates*)	especifica o período de validade do certificado, em formato 'data-hora'.

Tabela 3.2 - Certificados de autorização SDSI/SPKI

Considerando um certificado de autorização com o campo "sujeito" (S1) indicando um *threshold subject 2 of 3* (N1 N2 N3), neste caso, será necessário a autorização da delegação de 2 dos 3 sujeitos (N1, N2, N3) para que o certificado possa ter o direito especificado propagado à frente. Conseqüentemente, para a redução de certificados são necessários *k* dos *n* certificados dos sujeitos indicados em *S1* (no exemplo, 2 dos 3 indicados) para que os nomes sejam resolvidos. Após a resolução dos nomes e as interseções dos campos de autorização e de validade serem realizadas, o resultando será num único certificado gerado em favor de um dos sujeitos.

Assim, além do aspecto da tolerância à intrusão o uso de *thresold subjects* é benéfico também quando o objetivo é tolerância à faltas.

3.3.2 Verificação de autorização e autenticação

Quando desejar acessar um serviço, o cliente deve fornecer a cadeia de certificados de autorização; esta cadeia deve provar que o cliente está autorizado a acessar o serviço desejado. O guardião do serviço verifica se a cadeia fornecida é válida e confronta o certificado com a política na *ACL* do serviço – para verificar se o principal tem os direitos exigidos; caso ambas as verificações ocorram com sucesso a operação solicitada é executada.

A autenticação no *SDSI/SPKI* é relativamente simples e envolve apenas a comprovação de posse da chave privada – através da assinatura digital – correspondente à chave pública autorizada a realizar o acesso.

Passo	Entidade(s) envolvida(s)	Descrição da atividade executada no passo
I	P → S	O cliente solicita acesso a um recurso protegido (sem autenticação e nem autorização) para fazer uma operação (*op*).
II	S → P	O guardião devolve ao cliente um *challenge*, "prove a posse de credenciais para executar *op* no objeto protegido pela *ACL*"
III	P	O cliente usa um algoritmo para descobrir se há uma cadeia de certificados de autorização (linhas 0, 0' e 0'', na Figura 3.1) que lhe delegue permissão para executar a *op*.
IV	P → S	Se a cadeia existe, o cliente assina a requisição de *op* (linha 1, Figura 3.1) que é enviada junto com a cadeia de certificados apropriada ao guardião em um *response* ao *challenge* proposto (linha 2 da Figura 3.1).
V	S	O guardião verifica a assinatura da requisição de *op* (prova de autenticidade) e também se a cadeia de certificados provê as permissões necessárias (prova de autorização).
VI	S → P	Se todos os passos anteriores foram concluídos com sucesso *S* honra o acesso pleiteado por *P*.

Tabela 3.3 – Trocas entre cliente e servidor em acesso a um objeto protegido por SDSI/SPKI

Para ilustrar como a autenticação e autorização se relacionam, na Tabela 3.3 é descrito um cenário de trocas entre o cliente (*P*) e um servidor (*S*), necessárias para o acesso em um objeto protegido por uma *ACL SDSI/SPKI* (Figura 3.2).

```
(acl
    (entry
        (subject
            (name
                (public-key
                    (rsa-pkcs1-md5
                        (e #23#)
                        (n
                            |AMMgMuKpqK13pHMhC8kuxaSeCo+yt8TadcgnG8bEo+erdrSBveY3CMBkkZqr
                            M0St4KkmMuHMXhsp5FX71XBiVW1+JGCBLfI7hxWDZCxGTMgbR4Fk+ctyUxIv
                            3CQ93uYVkg9ca6awCxtS0EI7sLuEB+HKuOLjzTsH+Txw9NAHq4r|
                        )
                    )
                )
            <nome do sujeito>)
        )
        (tag
            (<application>
                (<op>)
                (<protocol>:<server>)
            )
        )
    )
 ...
)
```

Figura 3.2 – Exemplo de uma entrada numa ACL SPKI

Se o passo V da Tabela 3.3 for bem sucedido, o servidor pode emitir um certificado reduzido para o cliente. Ou seja, o servidor emite um único certificado com o conjunto de operações que o cliente pode executar no objeto protegido delegando os direitos diretamente do servidor para o cliente, sem a cadeia de intermediários.

Na próxima vez que o cliente solicitar uma operação no mesmo objeto de serviço,

não precisará refazer o passo III da Tabela 3.3, bastará apenas enviar o certificado recebido diretamente do servidor.

Na efetivação do controle de acesso (passo V da Tabela 3.3) o guardião executa um conjunto de verificações. A seguir será mostrado um exemplo das atividades envolvidas em tal controle:

1. O guardião verifica o *timestamp* da requisição para se certificar que a mensagem é recente.
2. O guardião cria o *tag* correspondente a requisição do cliente para confrontá-lo com o que foi devolvido no *challenge* (passo II - Tabela 3.3.).
3. O guardião verifica se a chave privada que assinou a requisição é o par da chave pública contida no campo sujeito da cadeia de certificados de autorização.
4. O guardião verifica a assinatura digital de cada certificado da seqüência de autorização, utilizando a chave pública contida em cada certificado verificado. Se todas as assinaturas forem autênticas e todos os períodos de validade dos certificados estiverem em vigência, então a seqüência é válida.
5. Se o passo 4 for bem sucedido, o guardião verifica se a seqüência de certificados de autorização concede permissões suficientes ao cliente para a execução da operação desejada. Caso afirmativo a solicitação é honrada.

A execução bem sucedida da etapa 3 (acima) representa a autenticação em *SPKI*. Se o passo 4 for bem sucedido significa que há um **caminho de confiança** (“uma seqüência de entidade confiáveis”) por onde foram propagados os certificados até o cliente. Este caminho de confiança tem uma função similar a hierarquia de *CA*s de *X.509*.

3.3.3 Premissass do SPKI e suas implicações

A seguir serão enumeradas brevemente, com base na RFC 2692 [41], as premissas que fundamentaram a especificação do SPKI:

1. O detentor de uma chave pública deve fornecer o mínimo de informações para conseguir suas permissões para executar uma ação. Isto é necessário no mínimo para que os certificados sejam anônimos (não identifiquem o nome do principal).

2. Os certificados *SPKI* devem ser úteis mesmo em ambientes com recursos limitados, como *smart cards* ou sistemas embutidos (*embedded systems*). Além disto, a estrutura do certificado deve ser simples e específica o suficiente para que o criador de um certificado tenha como maior preocupação a definição dos campos de tal estrutura e não mensagens de erro na leitura dos mesmos.
3. Não deve ser necessário bibliotecas de código para o empacotamento (*packing*) ou interpretação/avaliação (*parsing*) dos certificados; particularmente bibliotecas *ASN.1* devem ser evitadas. Além disto, na medida do possível o processo de *packing* e *parsing* não deve envolver funções recursivas por questão de eficiência.
4. O certificado deve ser assinado exatamente como será transmitido. Não deverá haver reformatação ou qualquer modificação no processo de verificação da assinatura.
5. As *CRLs* (*Certificate Revocation List*) devem carregar datas de validade explícitas. As datas em uma seqüência de *CRL*s não podem se sobrepor.

Estas premissas geraram implicações, citadas na RFC 2693 da *IETF* [42], que estão reproduzidas de forma reduzida a seguir:

1. A padronização de certificados *SDSI/SPKI* teve como objetivo principal a autorização ao invés da autenticação (para atender o item 1 das premissas).
2. Espaços de nomes são locais, mas as identificações são globais – através de chaves públicas (para atender o item 1 das premissas).
3. Os certificados de autorização associam permissões (autorização) a uma chave pública (autorizações ditas anônimas – para atender o item 1 das premissas). Os certificados de nomes associam nomes a chaves.
4. É adotado *S-expressions* [45] como o formato padrão para os certificados. Além disto, por razões de simplicidade e eficiência, foi definida uma forma canônica para as *S-expressions* dos certificados (para atender os itens de 2 a 4 das premissas).
5. As *CRL*s são datadas e publicadas com período de validade em locais previstos e alternativos. São adotados períodos de validade curtos para a delegação de direitos. Porém, uma vez concedido um direito, enquanto o mesmo for válido não será revogado. Em contrapartida, é prevista a revalidação *on-line* para dilatar o período de concessão de direitos quando assim for desejado (para atender o item 5 das premissas).

3.3.4 Considerações sobre o SDSI/SPKI

A infra-estrutura *SDSI/SPKI* permite a construção de relações de confiança de maneira descentralizada e democrática (principais escolhem em quem confiar). A descentralização propiciada pelas cadeias de confiança não implica na criação de pontos centralizadores que aumentam a vulnerabilidade do sistema ou diminuem a confiabilidade do mesmo. Além disto, aspectos como desempenho e escalabilidade devem ser favorecidos com as propriedades das cadeias de confiança *SDSI/SPKI*.

Outro aspecto importante sobre o *SDSI/SPKI*, embora não explorado neste texto, são as facilidades de interoperabilidade introduzidas em [42]; são previstas traduções de certificados de autorização de infra-estruturas conhecidas como *X.509*, *SSL* e *PGP* para certificados de autorização *SDSI/SPKI*.

De maneira geral, o *SDSI/SPKI* é uma solução potencialmente poderosa como uma configuração de controle de autenticação e autorização em sistemas distribuídos de larga escala. Porém, a dependência do conhecimento prévio das cadeias de certificados de autorização, necessárias para determinado acesso, caracteriza em certas circunstâncias um problema de difícil solução. Uma grande parte dos trabalhos sobre cadeias de confiança é dedicada à determinação de cadeias de certificados que viabilizem os acessos.

3.4 O problema da busca da cadeia de certificados

Neste item, são mostrados os esforços dos pesquisadores, presentes na literatura técnica, no sentido da busca de cadeias de certificados que levem a prova de autorização de acesso. Inicialmente é mostrada uma experiência de armazenamento dos certificados *SDSI/SPKI* no *DNS* (*Domain Name Server*). Em seguida são apresentadas buscas de cadeias de certificados de autorização interpretadas como grafos (distribuídos) e também, uma proposta de interpretação das cadeias de certificados como grupos. Por fim, são relatadas experiências do grupo do MIT, criador do *SDSI/SPKI*.

3.4.1 Armazenamento e recuperação de certificados SDSI/SPKI no DNS

A busca da cadeia de certificados de autorização comprovando que o usuário possui

permissão para um acesso, como comentado em [43, seção 6.3.4] é sempre responsabilidade do requerente (cliente).

Em [42, página 3] é exposto que: "*Se o detentor de uma chave desejar usar como repositório de certificados o LDAP, o servidor PGP ou a base do DNS, é critério seu e não é função do SPKI WG especificar*". Em [44, página 2] é previsto que padrões de serviços de nomes devem ser reconhecidos no *SDSI* versão 1.0.

Com base na citação do parágrafo anterior, em [46], o *DNS* foi usado para armazenar e recuperar os certificados *SDSI/SPKI* – Versão 1.0. Na proposta são utilizadas extensões acrescidas pela *RFC 2065*[12] ao *DNS* para permitir o armazenamento de registros de certificados. Além disto, são propostos algoritmos de buscas em avanço (*forward*) e em retrocesso (*backward*), que incluem a filtragem e a recuperação dos registros de certificados presentes no *DNS* e pertencentes à cadeia de interesse.

O autor sustenta sua escolha pelo *DNS* afirmando que o mesmo corresponde ao "serviço de nomes da Internet" e, portanto, é o local "natural" para o armazenamento dos certificados *SDSI/SPKI* no ambiente da rede mundial.

A arquitetura proposta em [46] comporta os tipos de certificados mostrados na Tabela 3.4. Além disto, a proposta introduz *Policy Administrations* (*PAs*) que administram certificados de permissão e *Trust Autorities* (*TAs*) ou *identity autorities*, emissores de certificados de identificação – similares as CAs do X.509, com a diferença que o principal escolhe em qual *TA* confiar.

Certificados	***Significado***
identidade (*Identity*)	especifica que um sujeito tem um certo nome no espaço de nomes do emissor. Este tipo de certificado pode ser utilizado também para expressar a crença do emissor que um sujeito pode prover a identificação de um serviço num outro espaço de nomes.
permissão (*Permission*)	expressa os direitos de um sujeito, concedidos pelo emissor.
delegação (*Delegation*)	autoriza restritamente o sujeito a emitir certificados em nome do emissor.
confiança (*Trust*)	autoriza de forma irrestrita o sujeito a emitir certificados em nome do emissor; indicando a total confiança do emissor no sujeito.

Tabela 3.4 - Certificados propostos em [46]

Assim, para ilustrar o funcionamento da arquitetura proposta, considere uma hierarquia de *DNS* simples, com duas companhias apenas (Figura 3.3): uma oferecendo um serviço (*service.un.org*) e a outra com um usuário (*department.acme.com*) desejando

utilizar o serviço. O servidor oferecendo o serviço deve conceder permissão de acesso ao usuário, efetivada através do seguinte fluxo de certificados de autorização (com respectivos tipos – entre parêntesis):

service.service.un.org (T) → *pa.security.un.org (D)* → *pa.department.acme.com (D)* → *user.department.acme.com (P)*

É também necessário que o cliente tenha assegurada a identificação correta do servidor. O usuário então emite um certificado buscando tal identificação em sua *TA* de confiança. Se a *TA* do principal não puder "certificar" tal serviço, delegará esta atribuição a uma outra *TA*, construindo assim uma cadeia de confiança até que a *TA* do serviço seja alcançada. Este fluxo de certificados (com respectivos tipos – entre parêntesis) e as entidades indicadas são mostrados a seguir.

user.department.acme.com (T) → *ta.department.acme.com (D)* → *ta.security.un.org (D)* → *service.service.un.org (I)*

Quando o usuário tentar o acesso ao serviço, o servidor deve iniciar a busca da cadeia de certificados que comprove a permissão necessária para o acesso desejado. Para que isto se concretize, o usuário necessita das *Policy Administrations* (*PAs: pa.security.un.org* e *pa.department.acme.com*). A busca se inicia com o algoritmo DFS em avanço (*forward*) pesquisando no *DNS* da companhia *un.org*, selecionando e colocando na seqüência correta todos os certificados a partir do serviço (*service.service.un.org*) até a indicação de que a cadeia continua no *DNS* da outra companhia (*pa.department.acme.com*).

Quando não há mais certificados no *DNS* da companhia *un.org*, o algoritmo de busca em retrocesso (*backward*) é acionado para localizar o restante da cadeia a partir de *user.department.acme.com* até *pa.department.acme.com*, sendo que se a cadeia não é encontrada o algoritmo falha. Caso a cadeia seja encontrada e o usuário passe pela autenticação, comprovando possuir a chave privada correspondente a sua chave pública, então o acesso é permitido. A Figura 3.4 esquematiza o fluxo de certificados para o exemplo citado.

[12] *Domain Name System Security Extensions, http://www.ietf.org/rfc/rfc2065.txt.*

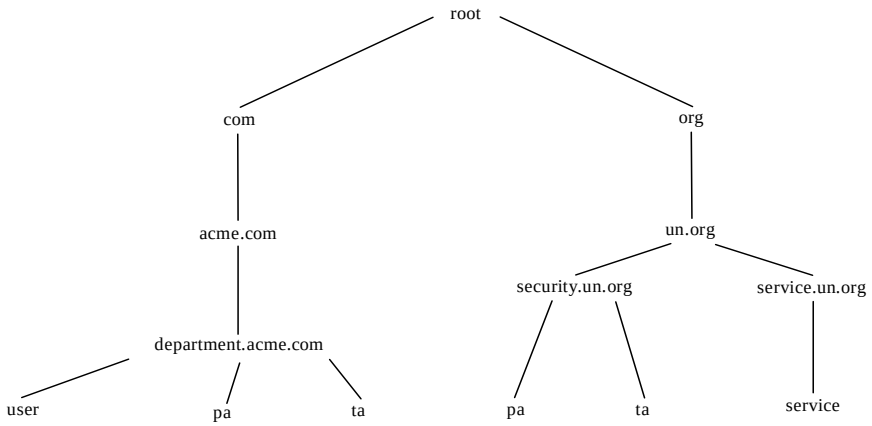


Figura 3.3 – Exemplo de armazenamento DNS para certificados SPKI, [47, pág. 9]

A busca da cadeia de certificados pode ser executada também pelo cliente. Neste caso, esse deve passar todos os certificados da cadeia para que o serviço os verifique junto com a solicitação da operação.

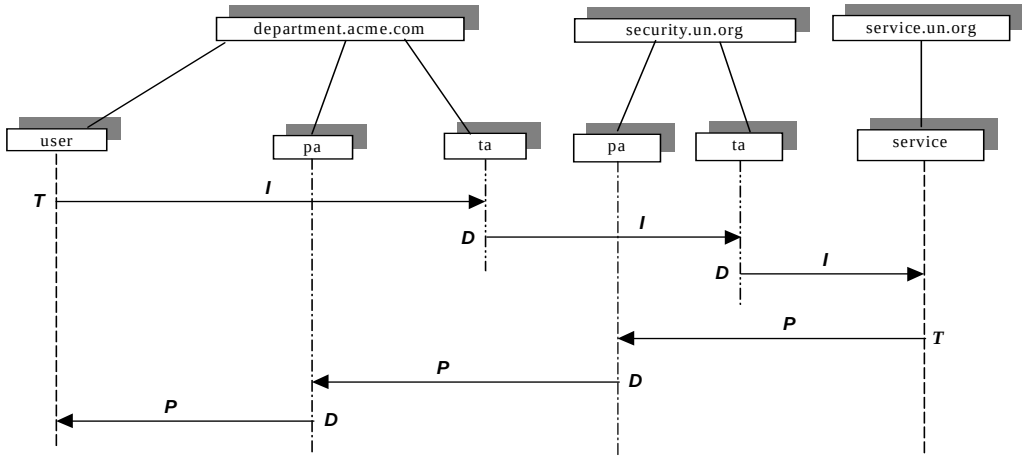


Figura 3.4 – Exemplo dos certificados SDSI/SPKI e locais de armazenamento DNS

A proposta em [46] pode ser interpretada como em desacordo com algumas das premissas do *SDSI/SPKI*: a solução está fundamentada no uso do *DNS* como repositório para os certificados "administrados" pela *TA* e *PA*. O uso da *TA* não parece ser compatível com a especificação de *SPKI*, porque esta desempenha um papel similar ao da *CA* de *X.509*. O papel da *PA* é menos incompatível com a especificação, mas desempenha uma função centralizadora na atribuição de permissões de acesso. Além disto, o *DNS* como repositório nas buscas de certificados mostra-se inadequado em recursos para filtragem dos certificados de interesse. Uma solução mais conveniente seria utilizar o *LDAP*

(*Lightweight Directory Access Protocol*), por exemplo, que permite filtragem na recuperação dos certificados. Adicionalmente, existe uma hierarquização no acesso a base de dados distribuída do *DNS*; um usuário comum não tem autoridade para alterar tal base de dados. Isto cria a dependência em relação à "autoridades locais" (administradores de sistemas) – o que vai contra a filosofia de *SDSI/SPKI*.

O uso do *DNS* pode ser considerado adequado no armazenamento e recuperação de certificados de identificação, uma vez que estes não sofrem alterações muito freqüentes. Porém, para fins de autorização o *DNS* certamente não parece um local muito adequado, mesmo porque há uma recomendação de que os certificados *SDSI/SPKI* tenham período de validade reduzido – já que não há mecanismo de revogação de uma autorização concedida. Logo, as alterações na base de dados do *DNS* seriam muito freqüentes e a dependência do administrador tornaria a solução muito limitada. Para minimizar o problema o autor comenta que o usuário poderia ter um "*DNS* pessoal" e ao mesmo tempo pertencer ao "*DNS* da companhia". A questão é: porque ter uma solução no *DNS* cheia de limitações? Não seria melhor uma solução mais adequada para armazenar certificados sem hierarquias?

Para finalizar pode-se comentar que este tipo de proposta, além de tudo, está um pouco superada porque a concepção de *SDSI* em sua versão 2.0 não mais menciona *special root* (como o *DNS*), isto era peculiar à versão 1.0 [43].

3.4.2 Busca nas cadeias de certificados interpretadas como grafos

Como comentado anteriormente, a concessão de permissões no *SDSI/SPKI* é feita através de certificados de autorização propagados por delegação de uma chave pública para outra. Considerando-se estes certificados numa base de dados [47], pode-se visualizar as cadeias formadas como um grafo direcionado (Figura 3.5, parte a).

O grafo (Figura 3.5) possui como *nós* as chaves emissoras dos certificados (k_x, x = 1,2,3,4,5,7) e os próprios certificados (c_y, y = 1,2,3,4,5,6,7), e como *arcos*, no sentido do fluxo de autorização, os direitos concedidos (mantidos entre chaves "{ }").

Na Figura 3.5 é possível observar também a indicação "(2,2)" identificando *um threshold certificate.* No caso, isto significa que ambos, k_2 e k_3, devem delegar o mesmo direito a uma outra chave para torná-lo válido; pode ser visto na Figura 3.5 (parte b) que

isto só acontece com o direito "r" sendo delegado à chave k_6, logo o único cliente de k_1, neste exemplo, é a chave k_6.

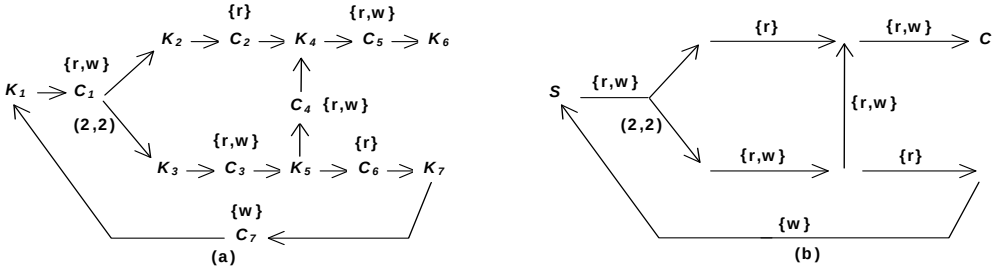


Figura 3.5 – Cadeia de delegação como grafo direcionado [47]

O autor argumenta que tanto o servidor quanto o cliente, possuindo números amplos de certificados, necessitam de um *gerente de certificados* (Figura 3.6). A tarefa de tal gerente dependerá do contexto associado, o gerente no lado do servidor além de gerenciar certificados do serviço associado deve tomar as decisões de autorização. A busca de cadeias de certificados de maneira rápida e eficiente para não comprometer o desempenho do sistema é a tarefa do gerenciador de certificados no lado do cliente.

Algoritmos rápidos no gerente se farão necessários tanto para as decisões de autorização como para as buscas de cadeias.

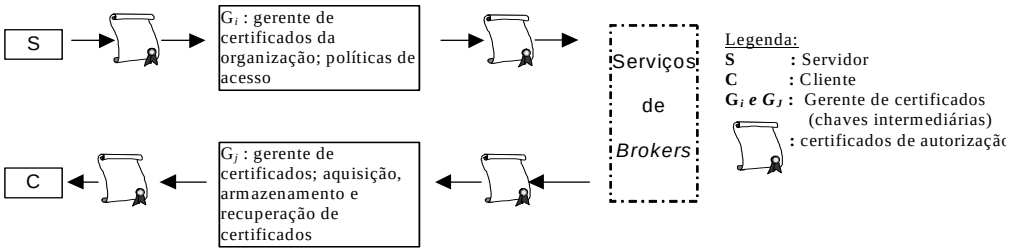


Figura 3.6 - Gerentes de Certificados

Antes de expor o funcionamento da procura nesta abordagem é necessário analisar o formato típico de uma base de certificados, segundo [47] esta geralmente se apresenta na forma de uma ampulheta (Figura 3.7). A forma se justifica pelo fato que entre números consideráveis de servidores e clientes há apenas alguns poucos níveis de entidades intermediárias (poucas chaves de confiança).

Retomando a idéia dos gerentes de certificados e tomando os mesmos como chaves intermediárias, a forma da ampulheta é confirmada. Assim, por exemplo, a atividade de

gestão de direitos de acesso aos servidores de uma organização provedora de serviços é atribuída a algum tipo de chave (gerente ou administrador de sistema) na organização. Similarmente, clientes concentram a atividade de aquisição de certificados em alguma entidade confiável em seu ambiente.

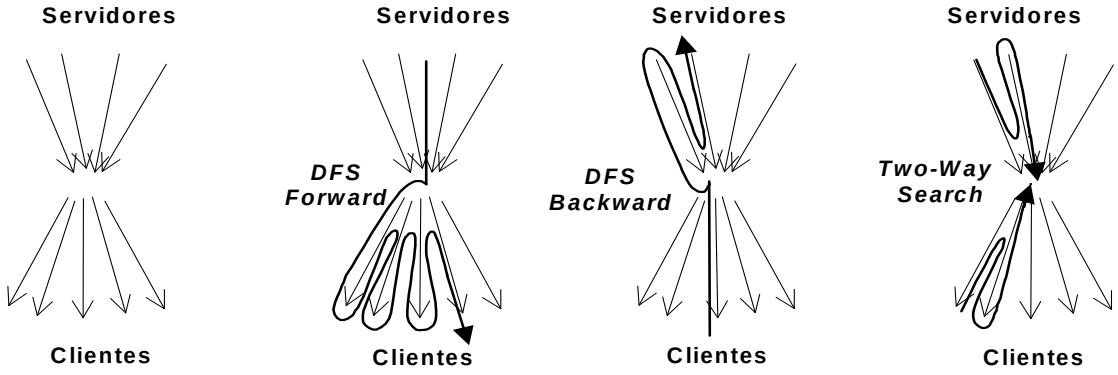


Figura 3.7 – Estratégias de busca com algoritmos DFS e two-way [47]

Os algoritmos apresentados em [47] são discutidos tomando uma base de certificados com apenas um intermediário (Figura 3.7). Estes algoritmos devem fazer decisões rápidas sobre a autorização do pedido e, basicamente, são divididos em busca em avanço (*forward*), busca em retrocesso (*backward*) e uma combinação de ambos (*two-way search*).

Quando aplicado ao grafo (representado os certificados de autorização), o algoritmo de busca em avanço (do servidor para o cliente, Figura 3.7) – com estratégia de busca em profundidade (*Depth-First Search – DFS*) – evita os *threshold certificate* (s*imple path search algoritm*), pois se os considerar poderá ser levado a execução com repetições cíclicas (*loops*).

Quando usado o algoritmo *DFS* em retrocesso (do cliente para o servidor – Figura 3.7), além da busca ser mais rápida e eficiente, não há restrições à utilização de *threshold certificates*. Porém, mesmo sabendo que o sentido de busca em retrocesso é vantajoso em relação à busca em avanço, uma conjunção de ambos (Figura 3.7) é proposta pelo autor como solução eficiente e com bom desempenho. Neste caso, as buscas se iniciam no servidor no sentido em avanço e no cliente em retrocesso; o encontro nesta busca composta se dá num ponto intermediário do caminho. Estes algoritmos são recomendados para fazer buscas em base de dados que podem ser amplas em número de certificados, mas

concentradas num único intermediário (base de certificados).

A proposta permite que se otimizem os algoritmos atuando em um único gerente, pois o autor pressupõe a base de certificados de permissão em forma de ampulheta. Porém, se as premissas assumidas não forem verificadas, os resultados podem não apresentar a mesma eficiência.

O autor apresenta uma proposta fortemente influenciada pela solução apresentada em [46], logo percebe-se também neste caso, a presença de agentes centralizadores (os gerentes) impondo uma administração localizada dos certificados. Além disto, como a proposta esta fundamentada na versão 1.0, nos certificados de autorização não são permitidos nomes *SPKI* como sujeitos, simplificando consideravelmente os algoritmos de busca.

Em [48] é relatada a experiência da implementação distribuída do *DFS* com busca em avanço, proposto em [47]. Nesta implementação se tem o primeiro trabalho com um algoritmo de busca da cadeia de certificados em ambiente distribuído. Diferente do imaginado, o autor considera que normalmente os certificados de um principal se encontram concentrados num único gerenciador de certificados e não distribuídos pelas bases da infra-estrutura de servidores da rede.

3.4.3 Busca nas cadeias de certificados interpretadas como grupos distribuídos

Em [49] é mostrado que os nomes locais *SDSI/SPKI* podem ser interpretados como grupos (conjunto de principais) distribuídos. A interpretação de nomes locais como grupos é baseada no fato que tais nomes podem ser resolvidos por múltiplos principais.

Na ótica de resolução de nomes, principais que resolvem (traduzem) um nome local em uma mesma chave pública são considerados membros de um mesmo grupo. Estes grupos – definidos tomando-se como base a resolução de nomes – são distribuídos ou, pelo menos, não envolvem uma hierarquia ou autoridade central que os gerencie.

A partir da interpretação de certificados de nomes como grupos, o autor lança mão de programação em lógica (*Prolog*) para mostrar que a resolução de tais nomes pode ser facilmente codificada em quatro regras de um programa em *Prolog*. Além disto, o autor mostra que estendendo a base de definições em lógica, é possível manipular certificados de autorização em buscas de cadeias de certificados mesmo com *threshold subjects*. A

solução do problema de busca, neste último caso, é alcançada com treze regras em *XSB* (uma variação de *Prolog*).

Na verdade, se avaliada a estratégia de solução empregada nos algoritmos de busca das cadeias de certificados, percebe-se que geralmente é empregada a busca em profundidade. Logo, a solução proposta para o algoritmo de busca de cadeias, resolução de nomes e manipulação de *threshold subjects* usando regras ou interpretação em lógica é na verdade passar a atribuição de execução da busca em profundidade ao ambiente de execução (Prolog/XSB). Evidentemente, o tratamento do problema baseado em lógica cria um nível de abstração que talvez facilite o entendimento e a resolução do problema. Porém, na implementação final do algoritmo de busca talvez isto crie dificuldades de otimização que podem prejudicar o desempenho de tal implementação.

3.4.4 Busca da cadeia de certificados na Proposta MIT

O grupo de pesquisa liderado pelo prof. *Ronald Rivest* (um dos idealizadores de *SDSI*) no *MIT*, considera que a descoberta da cadeia de certificados deve ser composta dos seguintes passos: seleção dos candidatos em potencial a compor a cadeia de certificados de autorização no conjunto de certificados armazenados no repositório, redução dos nomes no conjunto selecionado (seção 3.3.1, item A) e início da busca em profundidade (*DFS*) a partir do servidor até o cliente [50], seguida da redução dos certificados de autorização (seção 3.3.1, item B). Esta é a estratégia atualmente empregada pelos idealizadores de SDSI/SPKI no processamento da autorização.

3.4.5 Considerações sobre as estratégia de busca da cadeia de certificados

A seguir serão apresentadas algumas considerações e discussões sobre algumas situações colocadas na literatura técnica.

Quanto a necessidade de agentes (ou gerentes) desempenhando o papel de chaves intermediárias, a justificativa dos autores é que estas chaves são escolhidas e não são passagens obrigatórias como nas CAs do X.509.

Com relação às bases de certificados centralizadas em um único repositório pode-se julgar a inadequação desta proposição considerando sistemas distribuídos e suas

dimensões. A proposta do [46] desvia deste entendimento geral supondo certificados distribuídos em *PA*s distribuídos (chaves intermediárias e repositórios) nos ambientes do cliente e servidor, respectivamente.

Dos algoritmos apresentados, todos baseados em busca *DFS*, pode-se comentar que são eficientes considerando um determinado formato da base de certificados e a concentração dos mesmos no gerente de certificados (no lado cliente); se estas condições não forem encontradas não se pode prever o resultado alcançado. Em linhas gerais, ainda pode-se afirmar que quando a cadeia de certificados não é encontrada todos os algoritmos de busca apenas falham. Esta é uma de nossas motivações para a proposta que será apresentada no próximo capítulo, ou seja, quando os algoritmos de busca falham, o que fazer? Quando da falha dos algoritmos há duas situações a se considerar: o certificado que faria um elo de ligação na cadeia pode ter sido perdido ou pode estar de posse de outra chave e a cadeia realmente não existir. A seguir será proposto um método automático para criar a cadeia de autorização entre o cliente e o servidor desejado.

3.5 Considerações finais

Em linhas gerais pode-se considerar que o modelo de autenticação e autorização empregado no modelo *SDSI/SPKI* se apresenta mais abrangente que a gerência de confiança propostas no *PolicyMaker e no KeyNote.* Isto porque no *SDSI/SPKI* a prova de conformidade é alcançada com as cadeias de certificados e as políticas são necessariamente monotônicas.

No *SDSI/SPKI* existem vários trabalhos envolvidos com a descoberta de cadeias de certificados, assunto não explorado suficientemente em outras abordagens baseadas no conceito de gerência de confiança. Adicionalmente, ambientes *SDSI/SPKI* não são dependentes de nenhuma linguagem de especificação, quer seja para a política ou para as credenciais, sendo apenas utilizadas *S-expression* para a definição dos dados numa *ACL*.

Nos sistemas clássicos o controle de autenticação e de autorização é baseado em nomes e apresenta sempre alguma centralização, apresentando como conseqüência um ponto único de falhas e vulnerabilidade no sistema distribuído, o que pode representar também deficiência no desempenho do sistema como um todo.

O *SDSI/SPKI* é baseado em chaves e certificados apresentando-se como imune às

limitações dos sistemas baseados em nomes, além disto, o SPKI é mais flexível e escalável que os demais.

O *SDSI/SPKI* usa certificados com períodos de validade bem definidos, descaracterizando de certa forma a necessidade de *CRL*s descrevendo políticas negativas como nos outros modelos de controle de acesso.

4. Um modelo de autorização e autenticação baseado em cadeias de confiança para sistemas distribuídos

4.1 Introdução

Neste capítulo será apresentada uma proposta de extensão do modelo de autorização do SDSI/SPKI que facilita a publicação e localização de certificados de nome e de autorização através de uma entidade atendendo a um determinado propósito, a federação. É introduzido também um esquema que permite associação entre federações, com o objetivo de atender as necessidades dos usuários de um ambiente heterogêneo e distribuído.

O modelo proposto permite a criação de novas cadeias de autorização e negociação de concessão de direitos de acesso sem utilizar nenhuma entidade concentrando autoridade. Também não há nenhum tipo de estruturação das relações de confiança entre as entidades do modelo; os relacionamentos são criados de maneira arbitrária e cabe a cada principal a responsabilidade de administrar os seus direitos de acesso.

Será apresentada também uma heurística para navegar pelas entidades do ambiente distribuído visando localizar um determinado direito de acesso e um exemplo para ilustrar o funcionamento do esquema proposto.

4.2 Federação: extensão ao modelo de confiança SDSI/SPKI

Antes mesmo de introduzir a noção de federação se farão breves rememorações sobre

aspectos importantes a se considerar no sentido de facilitar o entendimento deste capítulo.

O fluxo de autorização do SDSI/SPKI, descrito no capítulo anterior, pode ser representado de maneira simplificada como é mostrado na Figura 4.1. O servidor *S* envia um certificado de autorização delegando direitos de acesso ao cliente *A* (C_{SA}), que o armazena em seu repositório local. Este mesmo cliente (*A*), repassa por delegação o certificado de autorização a um cliente *B* (C_{AB}), que também o armazena em seu repositório local.

Quando o cliente *B* desejar acessar o servidor *S* deverá assinar ("*A*( )") o pedido de acesso (*req*) e buscar a cadeia de certificados de autorização ($C_{SA} + C_{AB}$) em seu repositório local. Então, o cliente *B* deverá construir uma mensagem contendo a requisição assinada (*A*(*req*)) e a cadeia de autorização que deverá ser enviada ao servidor *S* ($C_{SA} + C_{AB} + A(req)$), Figura 4.1.

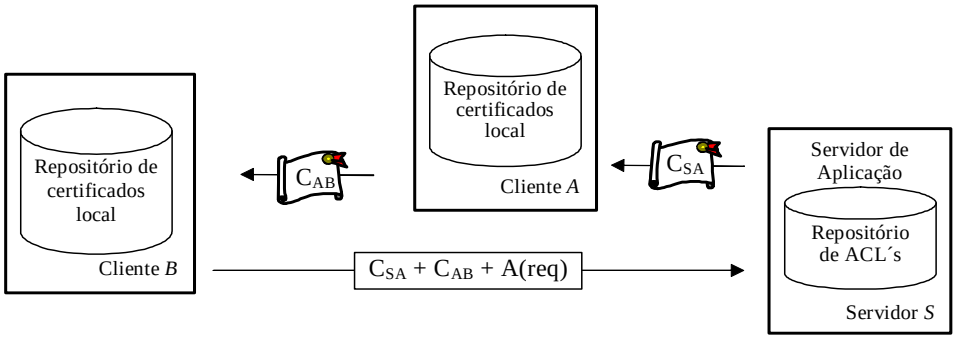


Figura 4.1 – Fluxo de autorização SDSI/SPKI

A gerência de confiança fundamentada na formação de cadeias de autorização do SDSI/SPKI permite observar que o modelo de confiança adotado neste sistema é fortemente focado no cliente. Outra conclusão é que as políticas de autorização estão distribuídas pela cadeia de confiança formada pela delegação de direitos, propagados através dos certificados de autorização de um servidor até um cliente.

Este esquema de autorização impõe restrições à localização de um determinado direito porque não oferece nenhum mecanismo através do qual seja possível buscar principais detentores de certificados com o direito de acesso desejado.

Na Figura 4.4 está sendo proposta a extensão do modelo de confiança do SDSI/SPKI mostrado na Figura 4.1, através da introdução da noção de *federação*. A **federação** é uma entidade que reúne principais com interesses afins e atua como um agente facilitador na

localização de certificados e principais, pois permite o compartilhamento do acesso ao seu repositório de certificados.

As cadeias de certificados SDSI/SPKI permitem dois tipos de leituras das relações de confiança resultantes. A cadeia de nomes ligados pode ser interpretada como uma *declaração de confiança numa identificação*, quando um principal redefine nomes de outros principais de seu relacionamento em seu espaço local de nomes. Do ponto de vista da autorização, a cadeia de nomes ligados serve meramente para fins administrativos – assim como a definição de grupos do SDSI/SPKI, pois no momento do acesso só chaves podem ser principais.

A cadeia de autorização é interpretada como uma *declaração de confiança numa autoridade* expressa através da delegação de direitos de acesso, porque a ação repassa autoridade ao sujeito do certificado. A integração da federação ao modelo de confiança do SDSI/SPKI estende a parte administrativa do mesmo, porque a federação não detém autoridade e nem participa das cadeias de autorização [51].

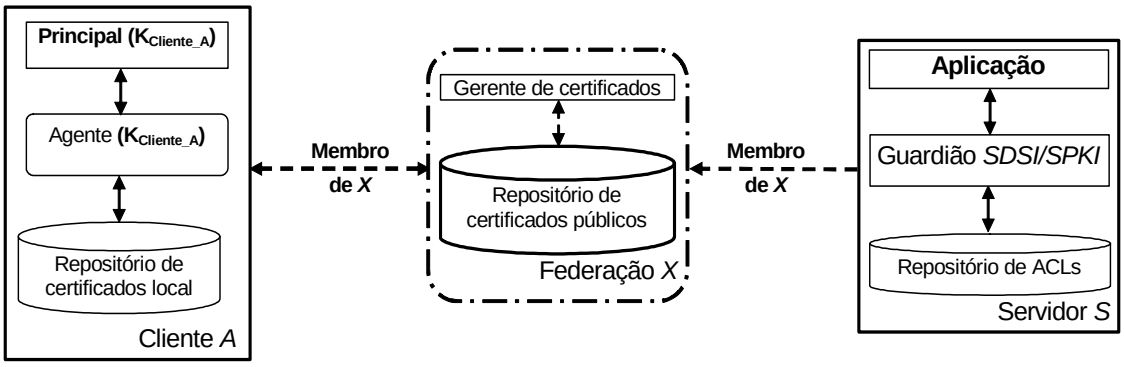


Figura 4.2 - Elementos da Federação

A seguir serão discutidos os elementos que compõem a federação (gerente de certificados, cliente e servidor) e seus inter-relacionamentos, como mostra a Figura 4.2.

4.2.1 Cliente

O cliente representa o principal que cria certificados de nome, propaga os certificados de autorização por delegação, participa de *threshold certificates* (processo de filiação, seção 4.2.3), emite requisições de acesso, participa da formação de novas cadeias (seção 4.4) e navega pela teia de federações (seção 4.5).

O armazenamento e a recuperação de certificados no espaço de nomes do cliente são concretizados a partir de seu agente. O agente ($K_{Cliente\_A}$, Figura 4.2) corresponde a um software: executando operações de gestão dos certificados disponíveis no repositório local, verificando e efetivando assinaturas digitais, pesquisando as cadeias de certificados, negociando a delegação de permissões, emitindo novos certificados de autorização, mantendo a consistência dos nomes locais e navegando pelas teias de federações. O agente está sempre ativo no sistema e o cliente se comunica com o mesmo através de um *binding* para a interface de operação do mesmo.

O principal motivo da criação do agente é a preocupação com a especificação do SDSI/SPKI, definindo que o uso do SPKI também deve ser possível em dispositivos com escassez de recursos e em dispositivos móveis como um celular, por exemplo. Neste caso, o agente poderia representar o cliente em uma plataforma de maior poder de processamento, por exemplo, com a qual o cliente se comunicaria através da interface de operação do agente. Além disto, o agente pode servir para automatizar atividades corriqueiras ou aborrecedoras que tomariam sem necessidade o tempo do principal (cliente).

4.2.2 Servidor de aplicação

O servidor de aplicação implementa os objetos de serviço, os quais protege com ACL's SPKI a partir de um guardião *SDSI/SPKI* (monitor de referência). O guardião *SDSI/SPKI* executa o controle de acesso aos objetos de serviço do servidor de aplicação. Todas as verificações descritas no passo V da tabela 3.3 são efetivadas pelo guardião.

Após o processo de redução de certificados o guardião pode emitir certificados de autorização ao cliente – esta ação representa uma nova cadeia de autorização. O guardião pode incluir a chave pública deste cliente em suas ACLs (do guardião) para agilizar futuros acessos do cliente. Para executar as delegações e negociações na propagação de permissões de acesso, o servidor também pode fazer uso de um agente similar ao que é definido no cliente (Figura 4.2).

Para efeito de auditoria são usados os registros de acessos (*log*s) das chaves públicas ao servidor. Quando necessário é executada a busca do certificado de nome na teia de

federações (seção 4.3) para identificar o principal correspondente à chave pública que efetivou o acesso; a partir de então pode-se fazer as devidas responsabilizações.

4.2.3 Gerente de Certificados

Para gerenciar o estabelecimento de relacionamentos de confiança (administrativos), entre os elementos da federação foi introduzido o gerente de certificados (GC). O gerente de certificados tem o objetivo de facilitar a interação entre o cliente e o servidor e efetivar a federação. Um gerente de certificados serve apenas ao grupo de principais de sua federação – as chaves públicas de seus integrantes formam um grupo SDSI. Um GC não participa ativamente de nenhuma cadeia de autorização; é principalmente um repositório de certificados de autorização SDSI/SPKI.

Para que um principal qualquer se filie a uma federação é necessário um endosso efetivado através da apresentação de um *threshold certificate* [52]. A assinatura do *threshold certificate* depende de *k-dentre-n* membros da federação. O número de membros *k* e *n* necessários no esquema de filiação são definidos por cada federação. Tal estratégia é utilizada para facilitar o processo de filiação no sentido de permitir que um subconjunto (*k*) de principais do conjunto (*n*)[13] possa atuar no papel do administrador da federação.

Na filiação de um principal, o seu certificado de nome é incluído no repositório da federação. Este certificado de nome será mantido pelo gerente de certificados para auxiliar no processo de identificação de principais (seção 4.2.4, item C).

emissor	*nome*	*sujeito*	*data de validade*	*Comentário*
$K_{federação\_X}$	Federação_X	$K_{cliente\_A}$	"data/hora"	"Membro da Federação *X* "

Figura 4.3 – Certificado de membro de federação (baseado no certificado da Tabela 4.1)

Para cada novo membro incluído na federação é emitido um "certificado de grupo" (certificado de nome expressando participação no grupo SDSI) para fins de comprovação da filiação (*membership*). Na Figura 4.3 pode-se ver um certificado de membro, enviando pela federação *X* ($K_{federação\_X}$) ao cliente A ($K_{cliente\_A}$), onde *Federação_X* identifica o grupo da federação *X*.

[13] O valor de *n* representa um conjunto de principais definidos pelo administrador da federação.

Portanto, ao gerente de certificados cabe a manutenção das informações referentes aos membros e associados de sua federação, removendo ou adicionando membros e associações com outras federações (seção 4.3) − sem promover conflito de interesses [53].

Operações de armazenamento e de recuperação de certificados de nome e de autorização estão disponíveis aos membros da federação através de interfaces padronizadas oferecidas pelo GC – que na busca de cadeias de certificados em seu repositório local se vale do algoritmo *Depth First Search* [54].

4.2.4 Estendendo o modelo administrativo SDSI/SPK

A partir da integração da federação ao modelo de confiança do SDSI/SPKI os clientes passam a ter uma alternativa a recorrer quando da falta de cadeias apropriadas para o acesso desejado. Ou seja, no modelo as federações podem servir como suporte para criação de novas cadeias.

No cenário da Figura 4.4, o servidor *S* envia um certificado de autorização delegando direitos de acesso ao cliente *A* (C_{SA}), que o armazena em seu repositório local. O cliente *A*, também armazena o certificado (C_{SA}) no repositório de certificados da Federação *X*.

Supondo que o cliente *B* não possua um direito de acesso ao servidor *S*, então o mesmo pode fazer uma busca na Federação *X*. Após receber o certificado (C_{SA}) da federação o cliente *B* pode pleitear junto ao cliente *A* a delegação de tal direito. Após conseguir o direito de acesso através do certificado C_{AB}, o cliente *B* pode fazer requisições de acesso ao servidor *S*, enviando a cadeia de certificados e a requisição assinada ($C_{AS} + C_{AB} + A(req)$).

A federação foi integrada ao modelo de confiança no sentido de facilitar a escalabilidade e auxiliar na resolução de nomes de principais SDSI/SPKI dentro de um sistema distribuído. A federação não participa das cadeias de autorização na delegação de direitos de acesso como uma chave, principalmente para não se tornar um ponto de concentração de autoridade e conseqüentemente de vulnerabilidade.

A seguir serão descritas as informações necessárias nos certificados de nomes e de autorização e nos tipos de certificados de autorização considerados na federação para efetivação da proposta. Todos os mecanismos de autenticação e autorização que serão mostrados nesta seção estão em conformidade com as especificações SDSI/SPKI.

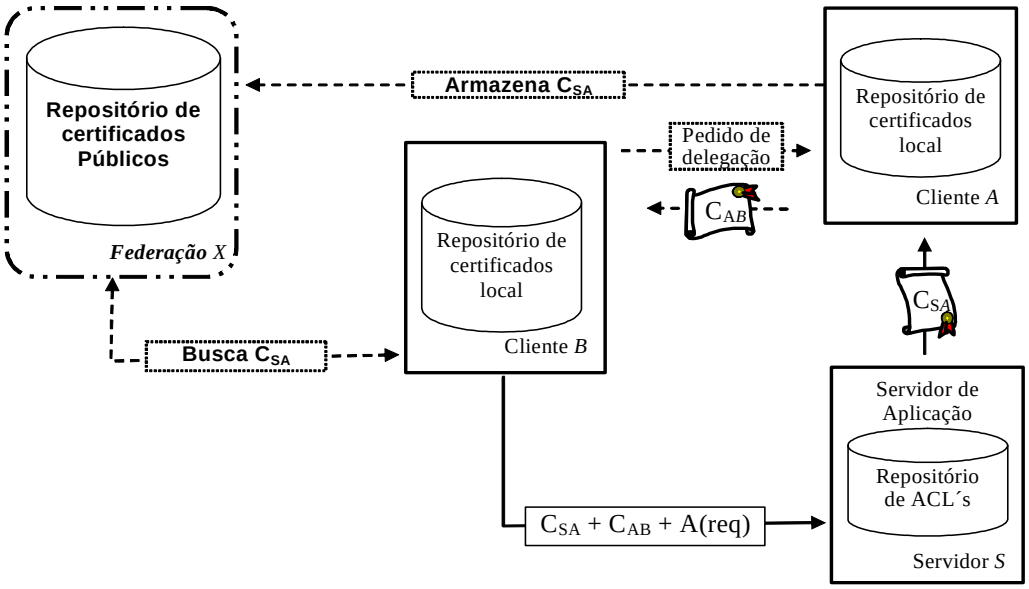


Figura 4.4 – Modelo de confiança estendido do SDSI/SPKI

A. Certificado de nomes no modo estendido

Nesta proposta os certificados de nome (Tabela 4.1) além de nomearem principais para facilitar o manuseio das chaves públicas, servem também para a definição do grupo de principais SPKI que formam a federação.

Campos	**Descrição**
Emissor (*Issuer*)	chave pública (ou hash da chave) da entidade (chave emissora) que está definindo o "Nome" no seu espaço local. O atributo opcional *uris* deste campo indica a *URI* onde pode ser encontrado o certificado de nome para a chave pública do emissor.
Nome (*Name*)	nome local que está sendo atribuído ao sujeito.
Sujeito (*Subject*)	identificação da entidade que está sendo definida (redefinida) no espaço de nomes local ao emissor.
Data de validade (*Validity dates*)	especificação do período de validade do certificado; em formato 'data-hora'.
Comentário (*Comment*)	campo opcional, adicionando texto informativo destinado a usuários humanos.

Tabela 4.1 - Certificado de nomes *SDSI/SPKI*, no formato estendido

Através da URI[14] (*Uniform Resource Identifier*) [55] informada no atributo *uris* do campo de chave pública do certificado de nome, pode-se facilmente estabelecer um contato com o emissor do mesmo – tanto se este for um gerente de certificados quanto se for um

[14] A URI fornece basicamente as seguintes informações: o mecanismo de acesso, a máquina hospedeira e o nome do recurso sendo acessado.

principal. Considerando-se a Internet, a URI pode ser uma URL (*Uniform Resource Locator*), por exemplo.

O campo de comentário foi adotado com o objetivo de permitir a descrição do objetivo ("*mission*") da federação, por exemplo. O campo de comentário poderá vir a ser utilizado pelo agente do cliente como um critério de busca de certificados na escolha de caminhos a percorrer durante a navegação pelas federações. O campo de comentário utilizado nesta proposta é um campo opcional na especificação do SDSI/SPKI; segundo a especificação pode-se utilizar este campo para qualquer fim.

B. Certificado de autorização no modo estendido

Para o certificado de autorização (Tabela 4.2), além dos campos tradicionalmente presentes neste tipo de certificado, adotaram-se os campos opcionais: localização do emissor (*issuer-loc*), localização do sujeito (*subject-loc*) e comentário. O objetivo do uso dos campos de localização do emissor e do sujeito foi facilitar o contato com estes principais, principalmente durante o processo de negociação de concessão de direitos de acesso (seção 4.4).

Campos	**Descrição**
Emissor (*Issuer*)	chave pública (ou *hash* dessa) da entidade que emitiu o certificado.
Sujeito (*Subject*)	chave pública (ou *hash* dessa), nome SDSI ou *threshold certificate* identificando a entidade que receberá a autorização.
Delegação (*Delegation*)	valor lógico (*True*/*False*), indicando se o sujeito pode ou não propagar a autorização que lhe foi delegada pelo emissor.
Autorização (*Authorization*)	campo que contém os direitos concedidos pelo emissor; representada como uma *S-expression*.
Validade (*Validity dates*)	especifica o período de validade do certificado; em formato 'data-hora'.
Localização do emissor (*issuer-loc*)	campo opcional especificando a URI do emissor do certificado.
Localização do sujeito (*subject-loc*)	campo opcional especificando a URI do sujeito do certificado.
Comentário (*Comment*)	campo opcional adicionando texto informativo, destinado a usuários humanos,

Tabela 4.2 - Certificados de autorização *SDSI/SPKI*, no formato estendido

O campo de comentário será utilizado nos certificados de autorização com fim análogo ao do certificado de nome. Assim, por exemplo, um certificado de autorização emitido por um servidor poderia descrever neste campo as aplicações e/ou recursos que o mesmo oferece.

Os certificados de autorização SDSI/SPKI podem ser utilizados para dois propósitos distintos. Em um, quando o *bit* de delegação estiver desligado (delegação não permitida) – os atributos de privilégio não podem ser repassados. Neste caso, o sujeito (principal) deve guardar o certificado de autorização como sendo do tipo *privado*, do qual só o próprio sujeito pode fazer uso. Quando o *bit* de delegação estiver ligado (delegação permitida) – o sujeito está de posse de um certificado de autorização do tipo *público*, do qual o mesmo pode fazer o uso que bem entender. Ou seja, o principal pode guardá-lo para seu uso particular, repassá-lo a terceiros na íntegra ou num subconjunto – quando isto se fizer necessário, ou ambos [27].

C. Autenticação baseada na cadeia de autorização

Como visto no capítulo anterior, na autenticação de principais SDSI/SPKI a identificação não é feita através de nomes, mas de chaves públicas e o mecanismo de autenticação é a assinatura digital. Assim, para que a assinatura digital seja verificada no destino a chave pública de um principal deve chegar de maneira segura até o sujeito. Como não há uma entidade distribuidora de chaves públicas certificadas no esquema SDSI/SPKI , as chaves públicas necessárias em uma autenticação são disponibilizadas através de uma cadeia de certificados de autorização.

A autenticação mútua no SDSI/SPKI é então conseguida com base em uma cadeia de autorização. O cliente ao fazer uma requisição a um servidor deve assiná-la e enviá-la junto com a cadeia de autorização que lhe concede os privilégios de acesso necessários. Quando da chegada da requisição ao guardião SDSI/SPKI a seqüência da cadeia de autorização será verificada. Se bem sucedida esta verificação, o guardião usa a última chave da cadeia de autorização (chave do cliente, presente no campo de sujeito da cadeia de autorização) para verificar a assinatura digital da requisição. Com esta assinatura conferida fica confirmada a autenticidade do cliente.

Todo certificado de autorização tem no campo do emissor a chave pública do principal que assina o certificado. Logo, para autenticar um servidor – sempre representado por uma chave pública, iniciando uma cadeia de autorização (primeiro certificado da cadeia de autorização), o cliente precisará do certificado de nome do servidor. O certificado de nome do servidor pode ser recuperado a partir de uma teia de federações (seção 4.3) ou buscado na URI do campo *issuer-loc*. Então, o cliente usa a chave pública

do certificado de nome para validar a autenticidade do servidor na cadeia de autorização. Se todas as verificações citadas forem bem sucedidas, a identidade do servidor estará garantida.

4.3 Teia de Federações

A filiação de clientes e servidores a federação facilita a localização de certificados de autorização, porém a federação geralmente tem a sua abrangência limitada ao fim a que se presta. Assim, os membros de uma federação necessitariam filiar-se a várias outras federações para alcançar certo nível de presença na rede. Este tipo de abordagem tem importantes limitações, pois um cliente não pode se filiar a todas as federações para ganhar abrangência global.

A proposta, então, é que as federações através de seus GCs (gerentes de certificados) se associem a outras federações criando relacionamentos mútuos de confiança que permitam ampliar o modelo de confiança administrativo formando teias de federações (Figura 4.5).

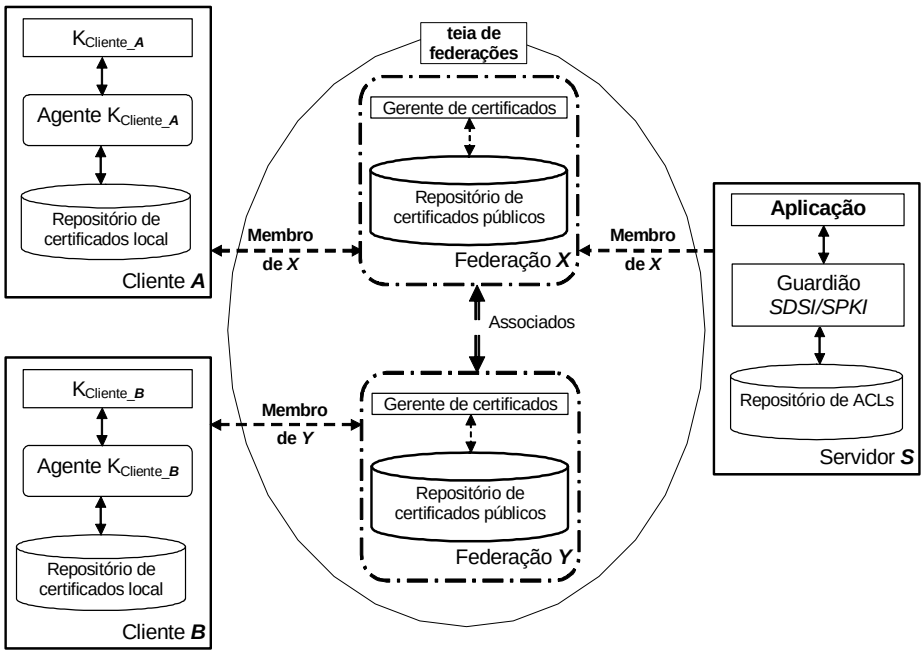


Figura 4.5 – Teia de federações

O estabelecimento de associações entre federações também é interpretado como admissão de novos membros nos "grupos SDSI" de cada federação envolvida. Só que neste caso, o novo membro (a outra federação) é tratado como um grupo definido e administrado em outro espaço de nomes – levando em conta a definição de grupos prevista no SDSI [43].

A Figura 4.6 mostra os certificados de membro emitidos pela federação *X* (4.1, 4.2 e 4.3) e pela federação Y (4.4 e 4.5). A associação mútua entre as federações *X* e Y é expressa através dos certificados 4.3 e 4.5, respectivamente.

	emissor	*nome*	*sujeito*	*data de validade*	*comentário*
(4.1)	$K_{federação\_X}$	Federação_X	$K_{cliente\_A}$	"data/hora"	"Membro da Federação *X*"
(4.2)	$K_{federação\_X}$	Federação_X	$K_{Servidor\_S}$	"data/hora"	"Membro da Federação *X*"
(4.3)	$K_{federação\_X}$	Federação_X	$K_{federação\_Y}$	"data/hora"	" Associado a Federação *X*"
(4.4)	$K_{federação\_Y}$	Federação_Y	$K_{Cliente\_B}$	"data/hora"	"Membro da Federação *Y*"
(4.5)	$K_{federação\_Y}$	Federação_Y	$K_{federação\_X}$	"data/hora"	"Associado a Federação *Y*"

Figura 4.6 – Certificados de membros de federações para o cenário da Figura 4.5

A teia de federações é formada arbitrariamente e cria caminhos de confiança entre as federações associadas de tal modo que um cliente pode fazer consultas (*queries*) a federações as quais não é filiado, bastando para isto que o GC da federação *home* (federação a que o cliente está diretamente ligado) seja associado à outra federação. Neste caso, o cliente deverá apresentar a federação associada o certificado de membro da federação *home* e o certificado de associação entre ambas (federação home e federação associada); estes certificados comprovam que há um caminho de confiança entre o cliente e a federação associada, através da federação home.

Com base na Figura 4.5 considera-se o cliente *B* desejando consultar o repositório da federação *X* para localizar o detentor de direitos de aceso ao servidor *S*. Então, o cliente *B* deve fornece a federação *X* o seu certificado de membro a federação *Y* (4.4) e o certificado de associação entre a federação *Y* e a federação *X* (4.5) para comprovar sua participação na teia de federações.

Se o cliente desejar acessar uma federação pertencente à teia, mas a federação *home* não está diretamente associada à mesma, o cliente deve reunir todos os certificados do caminho de associações (mútuas) entre as federações a partir da federação *home* até a federação de destino. Este procedimento permite a qualquer cliente da teia o acesso a

qualquer federação da mesma, bastando para isto descobrir o caminho (de confiança) entre ambos.

4.4 Criação de novas cadeias

Na literatura técnica científica são várias as experiências com a busca de cadeias de certificados SDSI/SPKI [46] [47] [48] [50]. Porém, em todas estas abordagens quando a cadeia de certificados não é encontrada a busca termina com uma exceção (falha) e o cliente fica impossibilitado de efetuar o acesso pleiteado. Neste trabalho, através da noção de federação, introduzida na seção 4.2, é criado um esquema que permite a um cliente localizar um certificado com a autorização desejada em uma teia de federações. Posteriormente, o cliente pode negociar com o possuidor do privilégio a concessão do mesmo para constituir uma cadeia de autorização possibilitando o acesso desejado.

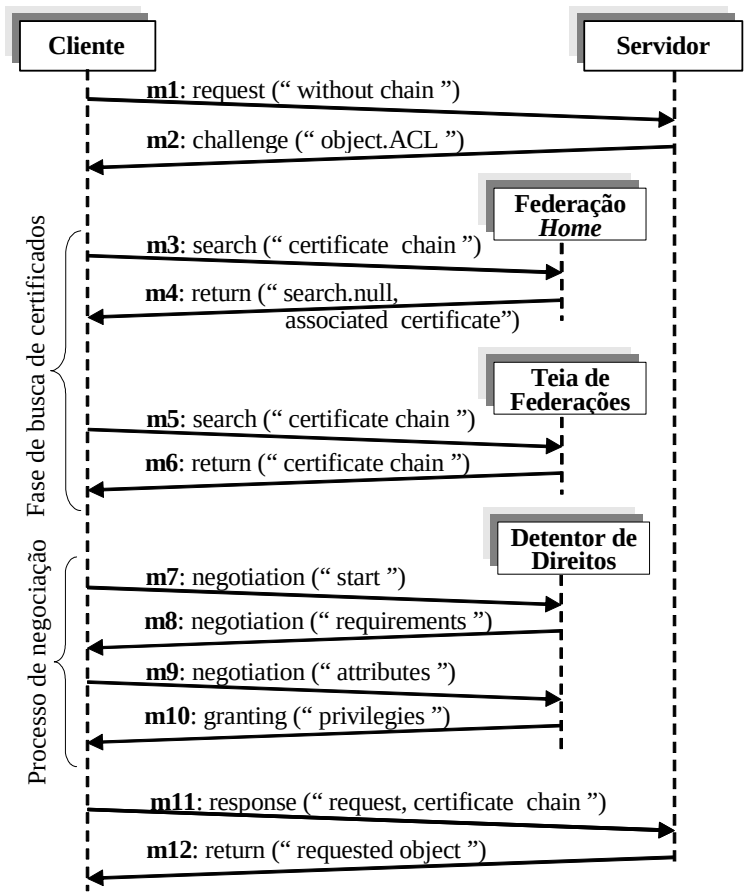


Figura 4.7 - Troca de mensagens para formação de novas cadeias de autorização

Para ilustrar o processo de formação de novas cadeias considera-se um exemplo de cenário como o mostrado na Figura 4.5. A Figura 4.7 mostra as mensagens trocadas entre o cliente, a federação *home*, a teia de federações e o detentor de direitos de acesso quando o cliente tenta acessar o servidor sem possuir a cadeia de certificados para fazê-lo.

O cliente envia um pedido de acesso ao servidor (mensagem m_1, na Figura 4.7). O servidor, por sua vez, envia um desafio em resposta a m_1. No desafio o servidor envia a ACL SDSI/SPKI protegendo o objeto solicitado e exige do cliente a comprovação da autorização para o acesso requerido (mensagem m_2). Neste caso, as informações da ACL SDSI/SPKI são úteis para acelerar o processo de procura.

Com as informações da ACL, o Cliente faz uma busca no repositório local por uma cadeia de autorização que o ligue ao servidor e permita o acesso desejado. A pesquisa deve retornar todas as cadeias de autorização que incluam a permissão necessária e tenha o servidor alvo como origem. Supondo que a busca local do exemplo resulte em insucesso, o cliente recorre a Federação *Home* encaminhando-lhe a pesquisa pelos certificados de autorização requeridos para o acesso ao servidor (mensagem m_3, na Figura 4.7).

Os critérios de busca considerados na pesquisa são as permissões requeridas e a chave pública do servidor. No caso considerado na Figura 4.7, esta pesquisa não resulta no retorno de nenhuma cadeia de autorização. Nesta situação, a Federação *Home* retorna ao cliente como resultado da pesquisa os certificados de membro nas federações associadas da teia de federações para que o mesmo possa prosseguir sua procura (mensagem m_4).

De posse dos certificados dos associados o cliente estende sua procura para outras federações da teia (mensagem m_5). A mensagem m_6 (Figura 4.7) representa o retorno da busca na teia de federações, ou seja, o momento em que o cliente recebe os certificados de autorização que procurava.

O cliente envia então ao detentor dos direitos o pedido de delegação dos mesmos (mensagem m_7, na Figura 4.7). A delegação de permissões a um solicitante pode ser concretizada de maneira simples – pelo fato do cliente e do detentor do direito compartilharem a mesma federação, por exemplo. Porém, dependendo da aplicação pode haver a necessidade de uma negociação mais complexa.

Na continuação do processo de negociação, após receber a mensagem m_7, o possuidor do direito requerido informa ao cliente um conjunto de requisitos para a concessão (mensagem m_8). O cliente reúne os requisitos exigidos e os envia ao detentor

dos direitos (mensagem m_9). Uma vez que os requisitos da aplicação foram atendidos o detentor dos direitos emite um certificado delegando as permissões ao cliente (mensagem m_{10}, Figura 4.7). Com esta última mensagem o processo de formação da cadeia é concluído e o cliente pode responder ao *challenge* proposto pelo servidor na mensagem m_{11} (Figura 4.7). Então o servidor atende a solicitação a partir da mensagem m_{12} [56].

4.5 Heurística de navegação em teias de federações

Quando se assume o ambiente distribuído sem nenhum mecanismo para dar suporte a localização de principais a navegação no ambiente torna-se uma atividade difícil. Nesta seção é mostrada uma alternativa para permitir a navegação (escolha dos principais formando o caminho que leva um cliente até o detentor de um direito) em um ambiente heterogêneo e distribuído sem o auxílio de um serviço de nomes.

O controle da navegação pela teia foi implementado no cliente, pois parece se adaptar mais facilmente ao modo de autorização do SDSI/SPKI. O cliente pode escolher os critérios de navegação que mais se adaptam ao seu tipo de perfil. Além disto, o cliente pode navegar pela teia e negociar a concessão de direitos de autorização sem ter sua identidade revelada. No caso, não é necessário o tradicional *account* e nem mesmo o cadastramento do cliente para que este tenha acesso a um servidor. O cliente negocia a concessão do direito de acesso ao servidor a partir de um terceiro para o qual o servidor a priori já delegou tal direito. O cliente mantém as estruturas de controle da navegação em seu poder, garantindo a sua privacidade com relação ao tipo de direito que negocia e com quem negocia.

Quando um cliente consulta um GC buscando direitos de acesso, este último responderá com um conjunto de certificados. Neste conjunto de certificados pode-se ter todos os certificados de autorização (*auth*) que contém os direitos pesquisados (se a busca resultar em sucesso) ou todos os certificados de associados às federações com as quais o GC em questão tem relações bilaterais de confiança (*associated-certs*).

A cadeia de confiança formada pela teia de federações dá o suporte básico para a navegação e assegura a autenticidade dos membros da teia, mas a cada consulta a um gerente de certificados (GC) de uma federação uma decisão do caminho a percorrer deve ser tomada. Com base numa estratégia de seleção do próximo principal a consultar,

apoiada por vários critérios que auxiliam na escolha, o cliente pode ir navegando nas diversas federações da teia (formada de maneira arbitrária) até encontrar um determinado certificado de autorização.

A seguir serão apresentados alguns critérios que estão sendo adotados no módulo do agente para auxiliar na seleção de principais/GCs para o caso citado anteriormente. Principais e GCs são tratados de maneira diferenciada, pois principais detêm direitos de acesso e GCs apenas auxiliam na localização de principais.

Algorithm 1 search (*R, k, auth*)

```
Require: R      // tabela de rank com todos os principais / associados
Require: F0     // certificado da federação Home
Require: k ∈ ℕ+
1   S0 = { F0 }
2   depth_max ← 3
3   i ← 0
4   while (i ≤ depth_max) do
5       G ← getCertsSubjects( Si )    //retorna o GC da federação de Si
6       while(G ≠ ∅) do
7           if ( (G ∩ R) ≠ ∅ ) then
8               P ← besthit ( (G ∩ R), k)
9           else
10              P ← criteriaSelection (G, k)
11          end if
12          G ← G \ P
13          while (P ≠ ∅) do
14              C ← firstElement (P, 1)
15              R ← R U C              // insere federação na tabela de rank
16              L ← searchAuthCerts (C, auth)
17          if (∀cert, cert ∈ L: cert.type = "principal") then
18              C.hit ← C.hit + 1
19              RightsNegotiation(R,L,k,auth)
20          else if (∀cert, cert ∈ L: cert.type = "associated-certs") then
21              Si+1 ← Si+1 U L
22              C.hit ← C.hit – 1
23              if (C.hit = 0) ∧ (C.fixed = false) then
24                  R ← R \ C          // remove federação da tabela de rank
25              end if
26          end if
27          P ← P \ C
28          end while
29      end while
30  i ← i + 1
31  end while
```

Figura 4.8 - Algoritmo de busca na teia de federações

Interpretando-se a teia de federações como um grafo, pode assumir que as estratégias de navegação podem ser: busca em amplitude (BFS – *Breadth-First Search*) ou busca em profundidade (DFS – *Depth-First Search*) [57]. A busca em profundidade não se aplica à

navegação na teia, pois quanto mais distante for o relacionamento entre o cliente e o principal detentor do direito, mais difícil deverá ser a negociação de concessão de direitos. Então, a estratégia adotada nesta proposta foi à busca em amplitude, BFS (linha 4, Figura 4.8).

A consulta a um GC pode resultar os certificados dos GC associados e nas consultas aos GCs associados o resultado pode ser outros certificados de GCs associados e assim por diante. Então, o agente utiliza como critério de parada um nível máximo de profundidade (*depth_max*, linha 2 da Figura 4.8) para não ficar indefinidamente procurando direitos que ninguém na teia de federações possui, por exemplo. A partir do GC *home* (nível 0) o cliente define (Figura 4.8, linha 3) para quantos níveis de profundidade vai executar o algoritmo BFS nas buscas em GCs associados.

Para a busca na teia de federações se assumiu que a probabilidade de sucesso em uma busca é maior para principais / GCs se a mesma for feita com estes últimos após já ter havido consultas com sucesso anteriormente. Esta estratégia está fundamentada na premissa que cada cliente, em geral, busca autorizações para fins similares – razão da formação da federação: atender a principais com interesses afins. Assim, o cliente em geral conseguirá nas mesmas federações ou com os mesmos principais os tipos de direitos desejados.

Para implementar esta estratégia de busca constroem-se tabelas onde cada linha da tabela contém um principal ou GC e um valor numérico indicando a taxa de *hit* (total geral de sucessos) em operações de procura com aquele principal ou GC. A taxa de hit é incrementada de 1 para cada sucesso e em caso contrário é decrementada de 1. Um valor de taxa de hit menor que 1 faz com o agente elimine o item da respectiva tabela. Estas tabelas são um histórico dos resultados bem sucedidos em transações entre o agente e o principal ou o GC em questão, atualizado dinamicamente e classificado por ordem decrescente de taxa de *hit* – tabelas com um *ranking* de *hit*s. A tabela de *rank* pode conter itens que não sejam atualizados dinamicamente e serão preenchidos manualmente pelo cliente. Neste caso, a taxa de *hit* daquele item não sofre atualização dinâmica; o GC home é inserido manualmente pelo cliente (linha 1, Figura 4.8) e possui uma alta taxa de hit (por *default*).

Toda vez que o agente receber um conjunto de certificados como resultado de uma consulta a um GC sua primeira ação é verificar se algum dos principais contidos nos

certificados pertence à respectiva tabela (Figura 4.8, linha 7). Em caso afirmativo, com base em seu *ranking*, o agente pode selecionar os *k* principais melhor classificados em tal tabela (Figura 4.8, linha 8), para os quais encaminhará a solicitação. A estratégia de limitar a busca a *k* GCs visa não inundar a teia de federações com muitas consultas simultâneas, dado que esta operação poderia ser feita em paralelo.

Sempre que a consulta à tabela de *ranking* não resultar nenhum item o agente utilizará outros critérios para auxiliar na seleção de GC (Figura 4.8, linha 10). Os critérios inicialmente considerados são: a localização geográfica (através do IP), o idioma de um país, o conteúdo desejado (através do campo de comentário) ou um sorteio.

O espaço de endereços do Ipv4 (*Internet protocol v4 address space*) mapeando a alocação geográfica de IP's no planeta[15], controlado pela IANA[16], serve de base para a localização geográfica. Com base na comparação da URI constante nos certificados de nome ou de autorização com tal mapa, o agente pode selecionar os certificados de GC / principal, respectivamente, de uma determinada localidade e então enviar uma mensagem aos escolhidos.

Um outro critério de seleção, análogo ao mapeamento de IP's, pode ser adotado para que o agente selecione certificados considerando o idioma falado em um determinado país.

Os certificados SPKI trazem um campo de comentário destinado ao preenchimento de informações que descrevem algo relevante sobre o emissor do mesmo (Tabela 4.1), assim o agente pode selecionar principais (escolher GC's associados) através da busca de palavras-chave ou expressões contidas no campo de comentário destes certificados.

Quando o cliente não deseja estabelecer nenhuma ordem de prioridade de escolha dos critérios citados acima, o mesmo pode optar pelo critério de sorteio, onde um GC é selecionado aleatoriamente.

Uma vez retornada a lista com os *k* GCs selecionados (Figura 4.8, linha 13), os mesmos serão inseridos um a um (linha 14, Figura 4.8) na tabela de *ranking* (Figura 4.8, linha 15) a medida que a busca (Figura 4.8, linha 16) é efetivada. Se o retorno da *query* for certificados de principais (linha 17, Figura 4.8) a taxa de hit é atualizada para aquele GC (linha 17, Figura 4.8) e o algoritmo de negociação é invocado (linha 19, Figura 4.8). Em

[15] (http://www.iana.org/assignments/ipv4-address-space)
[16] *Internet Assigned Numbers Authority*

caso contrário (Figura 4.8, linha 20) os certificados dos GCs associados retornados são anexados a lista de GCs pertencentes ao próximo nível do grafo (linha 21, Figura 4.8) e a taxa de hit para aquele GC é atualizada; se a taxa de hit for menor que 1, o GC será eliminado da tabela de ranking (Figura 4.8, linha 24).

Quando todos os GCs de um mesmo nível do BFS foram consultados e os direitos desejados não foram conseguidos, o nível do BFS é incrementado (linha 30, Figura 4.8) e o processo se repete até que o nível de profundidade seja alcançado ou a negociação resulte em sucesso.

Algorithm 2 RightsNegotiation (*R, L, k, auth*)

```
Require: R          //tabela de rank com todos os principais
Require: L          //certificados de autorização retornados pela busca
Require: k ∈ ℕ+
1    G ← getCertSubjects(L)    //retorna detentor de direitos
2    while (G ≠ ∅) do
3        if ( (G ∩ R) ≠ ∅ ) then
4            R_k ← besthit ( (G ∩ R), k)
5        else
6            R_k ← criteriaSelection (G, k)
7        end if
8        G ← G \ R_k
9        while (R_k ≠ ∅) do
10           X ← firstElement ( R_k, 1)
11           R ← R U X          //insere principal na tabela de rank
12           negotiationSucessuful ← negotiation (h, auth), h ∈ X
13           if (negotiationSucessuful = true) then
14               h.hit ← h.hit + 1
15               exit
16           else
17               h.hit ← h.hit – 1
18               if (h.hit = 0 ∧ h.fixed = false) then
19                   R ← R \ X    //remove principal da tabela de rank
20               end if
21           end if
22           R_k ← R_k \ X
23       end while
24   end while
```

Figura 4.9 – Algoritmo de negociação com os principais retornados pelo GC

A primeira parte do algoritmo de negociação (Figura 4.9, da linha 1 até a linha 10) funciona de maneira análoga ao algoritmo de busca (Figura 4.8, da linha 5 até a linha 14). A partir de então (linha 11, Figura 7) se a negociação resultar sucesso, a taxa de hit é atualizada (Figura 4.9, linha 14) e a busca termina (Figura 4.9, linha 15). Em caso contrário, além de atualizar a taxa de hit (Figura 4.9, linha 17) o algoritmo de negociação

verifica se o valor para a mesma é inferior a 1. Em caso afirmativo, o principal é removido da tabela de ranking (linha 19, Figura 4.9) e a busca é repetida até que a negociação logre êxito ou a lista de principais se esgote.

Uma explicação mais detalhada do funcionamento do algoritmo de navegação na federação será mostrada no cenário seguinte.

4.6 Cenário

A seguir será descrito um cenário que permitirá a navegação (comentada) pela teia de federações com base na heurística do algoritmo introduzido na Figura 4.8. Em seguida, é feita uma contextualização do esquema de navegação pela teia para mostrar um exemplo de utilização do mesmo em uma aplicação de vendas na Internet envolvendo um cliente, um servidor de aplicação (sítio de comércio eletrônico), um banco, uma teia de federações de instituições de cartão de crédito e seus respectivos representantes.

4.6.1 Navegação pela teia de federações

Os GCs da Figura 4.10 encontram-se mutuamente associados entre si. Cada GC agrupa os principais dos países indicados no nome da federação, formando uma teia de federações dispersas pelo planeta. Como regra de formação, observa-se que F_{BK_BR}, por exemplo, representa a federação do Banco *K* ($F_{\mathbf{BK}\_BR}$) localizada geograficamente no Brasil ($F_{\mathbf{BK\_BR}}$), implementada pelo gerente de certificados (GC_F_{BK_BR}). A federação tem como membros o cliente *B* e o Representante *BK* e como associados F_{CC_BR} e F_{CC_EUA}.

O GC_F_{BK_BR} detém os respectivos certificados de nome dos membros e dos associados e os certificados de grupo[17] dos associados. Os demais nomes de federações da Figura 4.10 têm o seguinte significado: tomando como exemplo a federação F_{CC_BR}, o termo F_{CC} identifica a federação das instituições de cartão de crédito de uma dada localidade geográfica, no caso o Brasil ($_{BR}$) e assim por diante [58].

Quando o cliente *B* tenta acesso direto – sem a cadeia de autorização, mensagem 1

[17] A designação “certificado de grupo” se faz aos certificados de nome que indicam participação numa federação (exemplos destes certificados podem ser encontrados na Figura 4.6)

da Figura 4.10 – ao servidor *S* (localizado geograficamente na China, continente asiático), esse lhe nega o acesso. Então, o servidor *S* devolve uma ACL informando que para conseguir tal direito o cliente precisa possuir uma cadeia de autorização a partir do representante da instituição de cartão de crédito (Representante *CC* – mensagem 2 da Figura 4.10). O cliente *B* através de seu agente, busca no repositório local o direito de acesso ao servidor *S* a partir do Representante *CC*. Como a busca resulta em falha, *B* inicia a navegação solicitando ao GC_F_{BK_BR} (GC *Home* – mensagem 3 da Figura 4.10) um certificado de autorização com tal direito de acesso (Figura 4.8, linha 16).

O GC *Home* não possui tal certificado (mensagem 4, Figura 4.10), então fornece os certificados das federações associadas (Figura 4.8, linha 21: L={GC_F_{CC_EUA}, GC_F_{CC_BR}}).

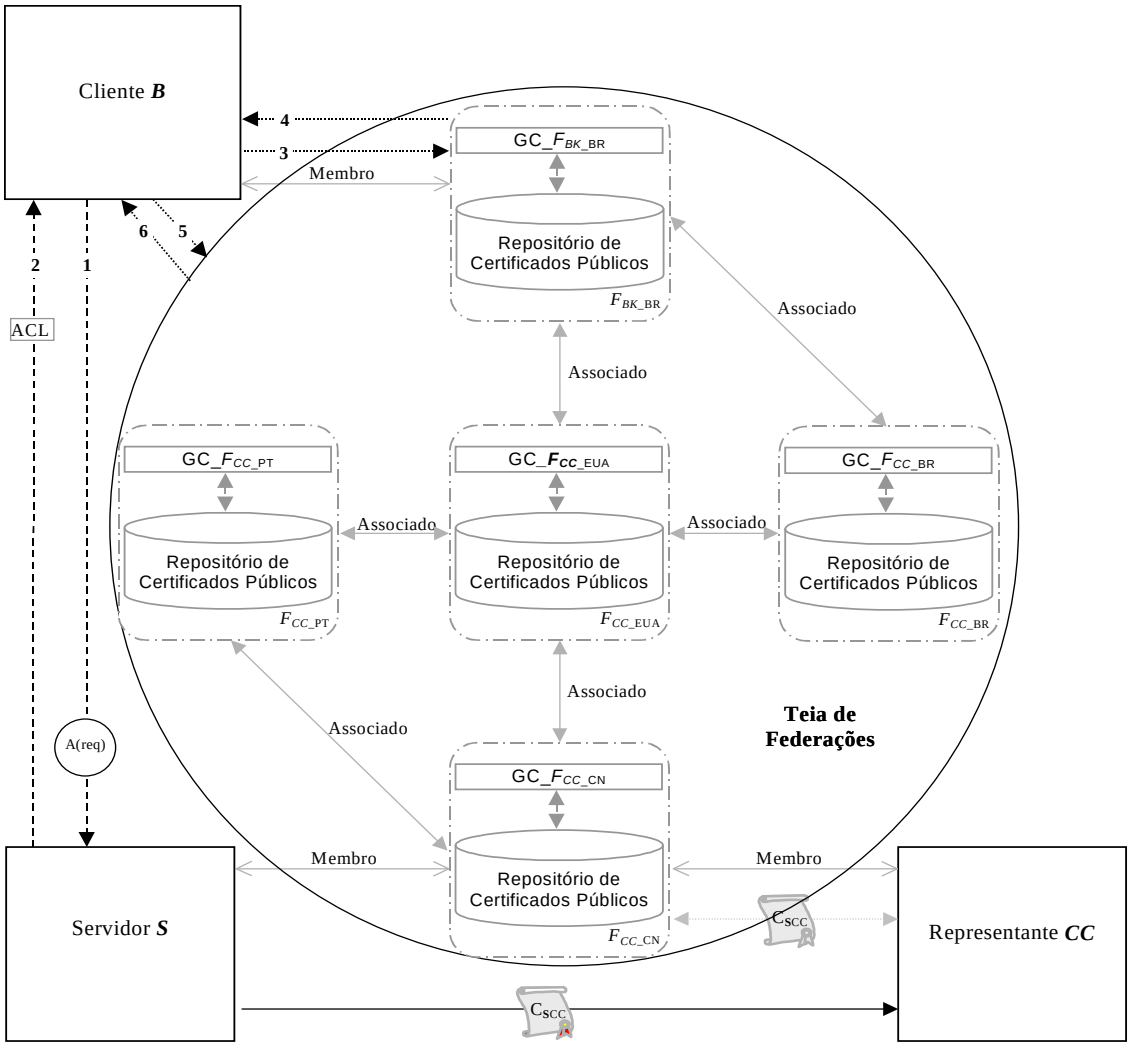


Figura 4.10 – Teia de federações de instituições financeiras dispersas pelo mundo

O cliente *B* executará uma busca por certificados de autorização nas federações associadas (mensagem 5, Figura 4.10). Após consultar sem sucesso a tabela de *ranking* (que está vazia – linha 7, Figura 4.8) o cliente tenta fazer uma seleção do GC *associado*, por exemplo, considerando primeiramente o critério IP, depois sorteio (linha 10, Figura 4.8). Não obtendo sucesso na seleção por IP, pois o servidor *S* se encontra na Ásia e os GCs (GC_F_{CC_EUA} e GC_F_{CC_BR}) não são deste continente, o próximo critério de seleção será o sorteio. Então, o cliente *B* escolhe aleatoriamente um dos certificados (GC_F_{CC_BR}, por exemplo). Após o sorteio, o cliente *B* insere o GC_F_{CC_BR} na tabela de *ranking* de associados (linha 15, Figura 4.8), inicializando o valor correspondente ao histórico para a taxa de *hit*, então faz a solicitação da autorização ao mesmo.

O GC_$F_{\boldsymbol{CC}_BR}$ não possui tal autorização, então devolve os certificados dos associados (GC_$F_{\boldsymbol{CC}_EUA}$ e GC_$F_{\boldsymbol{BK}_BR}$). O cliente *B* verifica que a busca fracassou e atualiza a taxa de *hit* na tabela, excluindo a entrada para o GC_$F_{\boldsymbol{CC}_BR}$ (linhas 21-24, Figura 4.8).

O cliente *B* com base no algoritmo BSF (BSF_depth = 1: GC_$F_{\boldsymbol{CC}_BR}$ e GC_$F_{\boldsymbol{CC}_EUA}$) seleciona o GC_$F_{\boldsymbol{CC}_EUA}$, ao invés de um dos associados devolvidos pelo GC_$F_{\boldsymbol{CC}_BR}$. Após inserir o GC_$F_{\boldsymbol{CC}_EUA}$ na tabela e inicializar o contador da taxa de *hit*, o cliente *B* faz a solicitação da autorização ao mesmo. A busca ocorre sem sucesso novamente e, portanto, o GC_$F_{\boldsymbol{CC}_EUA}$ devolve os certificados dos associados (GC_$F_{\boldsymbol{CC}_BR}$, GC_$F_{\boldsymbol{CC}_PT}$, GC_$F_{\boldsymbol{CC}_CN}$ e GC_$F_{\boldsymbol{BK}_BR}$).

Após atualizar a tabela de *ranking* e incrementar o nível de profundidade para o BFS (linha 30, Figura 4.8) o cliente *B*, com base no critério anteriormente estabelecido, seleciona entre os GCs com BFS_depth = 2 (GC_$F_{\boldsymbol{CC}_PT}$ e GC_$F_{\boldsymbol{CC}_CN}$) o GC_$F_{\boldsymbol{CC}_CN}$, pois o IP do servidor *S* está incluso no grupo de IPs da região do GC_$F_{\boldsymbol{CC}_CN}$, de acordo com o mapa de IPs sob responsabilidade da APNIC[18]. O cliente *B* faz a solicitação do certificado de autorização ao GC_$F_{\mathbf{CC}_CN}$ e obtém êxito, então o certificado é devolvido ao cliente *B* (mensagem 6, Figura 4.10).

Na próxima seção será descrito um exemplo de uma aplicação de compras pela Internet onde o cliente negocia a concessão de direitos de acesso e este cenário é

[18] (*Asia Pacific Network Information Centre* – http://www.apnic.net)

contextualizado. Porém observa-se que a aplicação onde seria possível utilizar o esquema proposto poderia ser uma outra qualquer.

4.6.2 Sitio de comércio eletrônico

Para facilitar o entendimento do cenário (seção 4.6.1) e visualizar o processo como um todo, considera-se que o servidor *S* da Figura 4.10 é um sitio de comércio eletrônico e que o cliente *B* carregou em seu navegador (*browser*) as páginas disponibilizadas pelo mesmo. Após navegar pelo sítio e selecionar itens, o cliente *B* escolhe a opção "ir ao caixa", que representa a mensagem 1, descrita na seção anterior. O servidor *S* na mensagem 2 (Figura 4.10) devolve a conta informando o valor da compra, a ACL informando o direito "liberar compra" e seu detentor (Representante *CC*).

O cliente *B* executa o conjunto de ações descritas na seção anterior (4.6.1) para localizar o certificado de autorização contendo o Representante *CC* como sujeito (que foi recuperado da F_{CC_CN}). A partir de então, o cliente *B* iniciará o processo de criação da nova cadeia de autorização que se finalizará com a concessão dos direitos de acesso (linha 19, Figura 4.8) – resultado da negociação com o Representante *CC*.

Para simplificar o processo e facilitar o entendimento presume-se que uma instituição administradora de cartões de crédito e um banco – com *CC* e *BK* como seus representantes, respectivamente – tenham algum tipo de parceria permitindo-lhe fácil compensação financeira. Com base nesta parceria o Representante *CC* delegou ao Representante *BK* o direito de "liberar compras" – no caso, o Representante *BK* pode liberar compras quando o pagamento deveria ser feito com cartões de crédito da instituição a que o Representante *CC* pertence. Como este tipo de "intermediação" do pagamento de compras (feita através do Representante *BK*) é uma operação especial, pois o normal seria o cliente pagar com seu cartão de crédito, o Representante *BK* não colocou este certificado no repositório de certificados públicos da federação a que pertence (F_{BK_BR}).

A Figura 4.11 resume todas as mensagens trocadas no exemplo de uma compra na Internet, sendo que a seguir será mostrado como acontece a fase de negociação em tal caso.

O cliente *B* quando recebeu o certificado de autorização da F_{CC_CN} (Figura 4.11, mensagem 6) recuperou do campo de localização do sujeito (*subject-loc*) a URI do sujeito

da delegação (Representante *CC*) e lhe dirigiu uma mensagem solicitando a delegação do direito "liberar compras". O Representante *CC*, devido ao acordo comentado anteriormente, devolve ao cliente *B* um certificado de autorização informando que delegou este direito ao Representante *BK*.

De posse do certificado o cliente *B* recupera a URI deste último e envia uma mensagem ao Representante *BK* solicitando a delegação do privilégio "liberar compras" a partir do servidor *S* (mensagem 7, Figura 4.11). O Representante *BK* identifica o cliente *B* como um cliente do Banco *K*, então informa ao cliente *B* que o requisito para a delegação do direito requerido é o pagamento do bilhete de compras, por exemplo, através de uma das opções: débito em conta corrente, boleto bancário ou outra qualquer (mensagem 8, Figura 4.11)

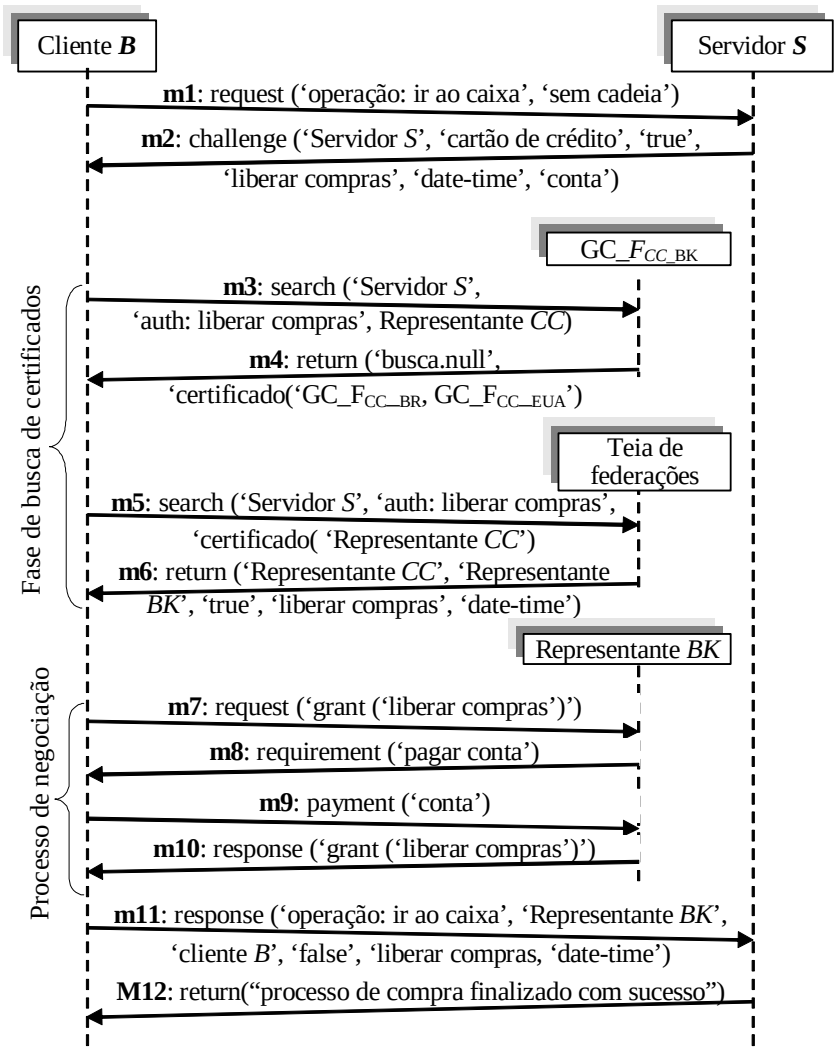


Figura 4.11 – Resumo das mensagens trocas no cenário de compras

O cliente executa o pagamento (mensagem 9) e na mensagem 10 (Figura 4.11) recebe a cadeia de certificados de autorização (certificado privado). De posse da cadeia o cliente a envia ao servidor *S* junto com a requisição assinada ("operação ir ao caixa" na mensagem 11 da Figura 4.11). O Guardião *SDSI/SPKI* faz as verificações comentadas no item V da Tabela 3.3 e se todas ocorrerem com sucesso o processo de compra é encerrado com sucesso.

Para efeito de controle das liberações de compras feitas pelo Representante *BK*, por exemplo, o Representante *CC* pode receber uma cópia dos bilhetes de compra pagos pelos clientes ao servidor *S* do sítio de comércio eletrônico e o elo de ligações entre os elementos do cenário então estará fechado.

No cenário descrito acima, não existia uma cadeia de autorização entre o Representante *CC* e o cliente *B*, porém, através da teia de federações foi possível criar a cadeia de autorização necessária para completar a operação de compra no sítio de vendas na Internet de maneira dinâmica e semi-automática.

Para o cenário descrito na Figura 4.11, pode-se considerar que não é necessária uma entrada contendo o cliente *B* nas ACLs do servidor de destino para autorizar o cliente a ter acesso ao mesmo. Logo, não é necessário o tradicional cadastramento de usuários (clientes) no servidor para que estes possam ter acesso ao mesmo. Conseqüentemente, todos os dados cadastrais dos clientes estarão armazenados apenas nas instituições com as quais o cliente tem um relacionamento mais direto.

No caso do exemplo acima o cliente poderia efetuar uma compra na Internet e pagar pela compra como se fosse cliente de uma administradora de cartões de crédito, apenas sendo cliente de um banco – tendo um relacionamento financeiro com tal administradora. Assim, não há números de cartão de crédito e nenhum outro dado pessoal do cliente circulando pela rede ou armazenado em base de dados de nenhum outro servidor que não seja o do banco onde o cliente tem conta corrente.

4.7 Considerações sobre a proposta

Como o gerente de certificados não figura nas cadeias de autorização como chave nas delegações sucessivas, o modelo proposto é totalmente descentralizado. Assim, o gerente não centraliza ou torna hierárquicas as relações de confiança entre cliente e

servidor e nem assume o papel de ponto crítico em relação à falhas e a vulnerabilidades ou ao desempenho do sistema.

Em aplicações da Internet o esquema proposto permite uma flexibilidade maior em aspectos referentes à legislação do que a infra-estrutura X.509, por exemplo. O cliente geralmente negociará a concessão de privilégios com um principal de seus domínios (da mesma localidade – país, por exemplo), este principal por sua vez poderá estar inserido em domínios remotos, então, haverá fortes vínculos entre o cliente e o principal local e entre este principal local e os principais dos domínios remotos. Assim, arbitrariamente, se definirão contextos de abrangência compatíveis com os universos onde cada principal atua.

A adoção da teia de federações isenta o servidor do gerenciamento de usuários e também isenta o cliente da necessidade da tradicional criação de uma conta (*account*) para ter acesso a um servidor – mesmo num contexto global.

O modelo proposto apresenta um suporte à gerência de certificados que permite a criação de novas cadeias de autorização; esta característica não é identificada em nenhum dos trabalhos relacionados. O esquema proposto é bastante flexível e automático, mesmo considerando que em alguns casos o número de trocas necessárias para criar a cadeia possa ser expressivo.

O fluxo de autorização, baseado na delegação de direitos de acesso adotado no SDSI/SPKI, aliado às extensões propostas neste trabalho, permitem observar que as políticas de autorização SDSI/SPKI estão distribuídas entre os membros da teia de federações e não concentradas nas ACLs do servidor de aplicação como nos sistemas clássicos.

4.8 Considerações finais

Neste capítulo foi apresentada a proposta que estende o modelo do SDSI/SPKI, possibilitando a busca distribuída de certificados de autorização através de teias de federações. Quando a cadeia de autorização não existe é possível criar novas cadeias através do processo de negociação.

Foi apresentado também um exemplo de aplicação para mostrar o funcionamento do algoritmo de navegação na teia e do processo de negociação de direitos de acesso.

5. Integração CORBASec – SDSI/SPKI

5.1 Introdução

As cadeias de certificados de autorização geralmente possuem uma abrangência localizada. No modelo introduzido anteriormente (capítulo 4) ampliou-se este alcance para o âmbito global. As redes de larga escala integram os mais variados componentes de *software* e *hardware*; no protótipo as opções pela adoção de padrões e pela tecnologia de middleware ofereceram a interoperabilidade e a uniformidade desejada neste ambiente. Através da introdução de conceitos e esquemas, em conjunto com as tecnologias de *middleware*, foi possível a efetivação do ambiente desejado.

Neste trabalho foi utilizada a programação distribuída baseada na plataforma CORBA [81] de objetos distribuídos. Tal opção pelo *middleware* não significa que o modelo não possa ser utilizado em outro tipo diferente de aplicação, por exemplo, pode-se trabalhar com objetos *web* sobre o *http*. Isto também não limita o modelo proposto a este estilo de programação.

O restante do capítulo está estruturado do seguinte modo: inicialmente são considerados aspectos relevantes do CORBASec (seção 2). Na seção 3, é mostrada a arquitetura do protótipo. Na seção 4, são discutidos os aspectos da implementação do protótipo. Na seção 5, a aplicação de teste é apresentada e na seção 6 são feitas conclusões.

5.2 Serviço de segurança do CORBA (CORBASec)

A especificação CORBASec [66] foi projetada para atender requisitos de segurança em sistemas distribuídos. O modelo CORBA de segurança é especificado na forma de objetos de serviço (COSS), provendo segurança para as informações e aplicações expressas como objetos distribuídos. O CORBASec define um modelo capaz de fornecer funcionalidades como: identificação e autenticação de principais, controle de acesso na invocação de um método remoto, comunicação segura (incluindo cifragem para garantir a confidencialidade bem como a integridade), não-repudiação, auditoria e administração.

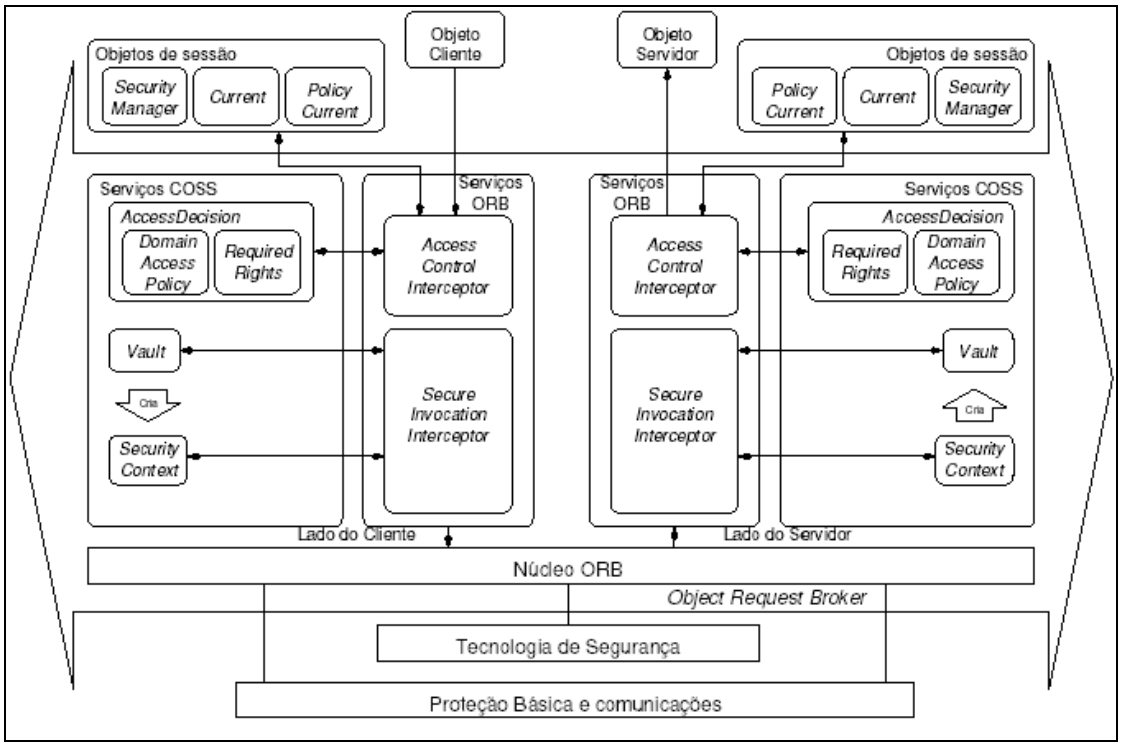


Figura 5.1 - Modelo estrutural do CORBASec [15]

No modelo CORBA de segurança os objetos são relacionados em quatro níveis (Figura 5.1): nível de aplicação – compreendendo os objetos de aplicação; nível de *middleware* – composto pelos objetos de serviço, serviços ORB e núcleo do ORB; nível de tecnologia de segurança – formado pelos serviços de segurança subjacentes e o nível de proteção básica – composta por uma combinação de hardware e sistema operacional da máquina local. O anexo V aborda com mais detalhes a arquitetura CORBA.

O CORBASec provê interfaces padronizadas que podem ser utilizadas com diferentes tecnologias de segurança: Kerberos (Anexo II), SPKM [59], SESAME [60] e SSL [61]. A escolha de uma tecnologia de segurança limita os serviços CORBA a esta tecnologia.

Um principal para agir no sistema deve primeiro estabelecer seu contexto ativando seus direitos para então poder acessar objetos. O objeto *PrincipalAuthenticator* implementa o serviço de autenticação no CORBASec, ativando credenciais no contexto do principal. O nível 2 de segurança (*SecurityLevel* 2 – *SL2*), realizado no âmbito da aplicação, permite que o principal mude os seus privilégios nas credenciais e escolha quais credenciais serão utilizadas em cada invocação.

O controle de acesso no nível 1 de segurança (*SecurityLevel* 1 – *SL1*) é realizado no ORB de modo transparente para as aplicações. Durante uma invocação ou em tempo de conexão (*bind time*) o interceptador de controle de acesso aciona o objeto *AccessDecision* que através do método *access_allowed* determina se a invocação do cliente deverá ser permitida ou não. Para isto é feito um confronto dos direitos fornecidos pelo cliente – através do objeto *Credentials*, com os direitos necessários (objeto *RequiredRights*) para acessar um recurso/objeto do sistema.

Além dos interceptadores de controle de acesso que atuam durante uma invocação, o modelo CORBASec também define os interceptadores de chamada segura − destinados as interceptações de baixo nível no sentido de garantir as propriedades de integridade e de confidencialidade nas transferências de mensagens correspondentes à invocação nas associações seguras.

A forma como é realizado o controle de acesso no CORBASec impõe ao modelo certas restrições de granularidade. O controle de acesso consiste basicamente no confronto dos atributos de privilégio (objeto *DomainAccessPolicy*) com os direitos requeridos (objeto *RequiredRights*). No objeto *DomainAccessPolicy* são definidos os direitos que refletem as credenciais do principal dentro de um determinado domínio – atribuídas ao mesmo quando ingressa no sistema de objetos do CORBASec. O objeto *RequiredRights* define os direitos necessários para que a invocação a um determinado método a partir de uma interface seja honrada.

Na Figura 5.2, aos atributos de privilégio “*AccessID:MELLO*” e “*AccessID:JOÃO*” são atribuídos os direitos de acesso “g” (“*get*”) sobre o domínio “Contas”. No objeto

RequiredRights é definido que o direito “g” é necessário para invocação do método “*verificaSaldo*” da interface “MembroDaConta”. Desta forma, principais dentro do domínio “Contas” que tenham um atributo de privilégio com o direito “g” poderão invocar o método “verificaSaldo” na interface “MembroDaConta”.

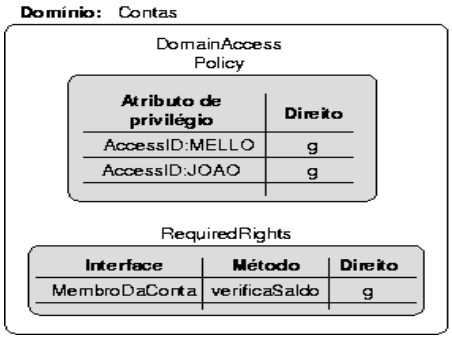


Figura 5.2 – Exemplo de controle de acesso no CORBASec [68]

Neste caso, nada impede que principais autorizados acessem todas as contas do domínio – no que se refere ao método “verificaSaldo”. Ou seja, todas as contas pertencentes ao domínio “Contas” estariam acessíveis a qualquer principal pertencente ao mesmo domínio e possuindo o atributo “g”. Tal situação ocorre porque as credenciais disponíveis visam o domínio e os direitos requeridos visam os métodos ocasionando um problema de granularidade. Em [61] são propostas algumas abordagens para o problema da granularidade:

1. Criar domínios separados para cada cliente. Isso permitiria que o controle de acesso fosse realizado completamente pelo CORBASec. Porém, esta abordagem não é escalável e modificações nas políticas de segurança geram modificações em cada domínio.
2. Impor um controle de autorização na lógica das implementações dos objetos. Desta maneira um único domínio conteria todos os objetos e não se teria mais os problemas relacionados a escalabilidade. Neste caso, a responsabilidade pela implementação da autorização passaria para a aplicação. Porém, qualquer alteração das políticas de segurança implicaria na atualização dos objetos de aplicação.
3. Introduzir um servidor de autorização, como por exemplo, o RAD (*Resource Access Decision*) [62]. Assim, os objetos repassariam a este servidor (RAD) as verificações de autorização. Com a centralização lógica das regras de autorização oferecida pelo

RAD ganha-se facilidades para modificar as políticas de segurança sem afetar as aplicações. Contudo, por se tratar de uma abordagem centralizada, a solução estará sujeita aos problemas clássicos de escalabilidade, desempenho e disponibilidade.

A integração do SDSI/SPKI ao CORBASec confere a este último uma alternativa de controle de granularidade fina através dos certificados de autorização, como será visto na seção 5.4.

5.3 Arquitetura do protótipo

Com o intuito de validar o suporte de autenticação e autorização proposto no capítulo 4, uma arquitetura genérica atendendo as especificações do modelo proposto e os requisitos específicos de um sistema de larga escala foram definidos, como se pode ver na Figura 5.3.

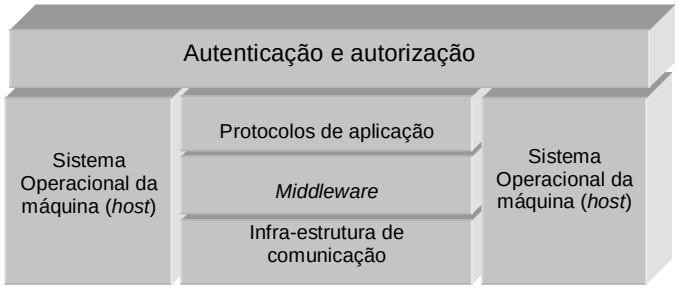


Figura 5.3 – Arquitetura genérica do protótipo

As opções de escolha tecnológica para o preenchimento de cada "camada" da arquitetura genérica foram fortemente influenciadas pelo ambiente Internet e se basearam unicamente no critério de tendência de utilização mais freqüente de cada item escolhido − isto não significa que outras alternativas não poderiam ter sido utilizadas. O protótipo é resultado de uma integração baseada em tecnologias padrão, de domínio público e componentes de prateleira (*Commercial Off-The-Shelf - COTS*) [63].

A Figura 5.4 mostra as tecnologias escolhidas para compor a arquitetura genérica, onde os códigos dos clientes e dos servidores são interpretados por máquinas virtuais Java (JVM – *Java Virtual Machine*) [14]. O Java foi adotado no protótipo devido a sua ampla utilização em aplicações distribuídas, principalmente, devido a facilidades no desenvolvimento de aplicações de uso amigáveis a partir de protocolos de aplicação na

Internet. Outro motivo da escolha da plataforma Java foram às restrições impostas pelas demais ferramentas utilizadas no protótipo – uma vez que as mesmas foram desenvolvidas no ambiente Java. No protótipo foi utilizada a ferramenta J2SE (Java 2 Plataform Standard Edition) versão 1.4.1.

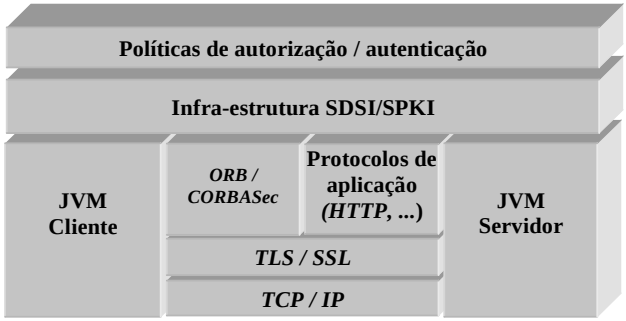


Figura 5.4 – Arquitetura do protótipo

O CORBA e o CORBASec fornecem facilidades para a interoperabilidade de aplicações distribuídas construídas a partir de sistemas heterogêneos. A proteção das mensagens em trânsito foi garantida pela camada TLS/SSL (*Transport Layer Security / Secure Sockets Layer*) apoiada na arquitetura TCP/IP. O TLS/SSL oferece as garantias de confidencialidade e de integridade na comunicação em rede entre componentes situados em máquinas físicas diferentes. No protótipo foi utilizado o ORBacus v3.3.4 [65] juntamente com o ORBAsec SL2 v2.1.6 [64]. O ORBAsec SL2 implementa somente alguns objetos do nível 2 da especificação CORBASec, mas esses já satisfazem as necessidades do protótipo. Como módulo SSL junto ao ORB foi utilizado o iSaSiLk [66], uma implementação do SSL desenvolvida pela Universidade de Graz, na Áustria.

O suporte a infra-estrutura SDSI/SPKI foi conseguido através da biblioteca JSDSI 2.0 implementada por [67], acrescida de algumas modificações. A rigor, a infra-estrutura SDSI/SPKI e as políticas de autorização são totalmente independentes de tecnologia.

5.4 Implementação do Protótipo

O modelo de integração proposto é parte de um projeto maior que visa o suporte de autenticação e autorização para a distribuição de documentos na Internet[19]. Tal modelo

[19] (http://www.das.ufsc.br/seguranca/)

integra o SDSI/SPKI ao modelo de segurança do CORBA (CORBASec) propiciando um controle descentralizado e de granularidade fina [68].

5.4.1 Modelo de integração SDSI/SPKI e CORBASec

O protótipo implementado é ilustrado na Figura 5.5. O modelo de integração proposto usa o nível de segurança 2 (SL2) do CORBASec que implica em controles de segurança em nível de aplicação, o que é totalmente compatível com o SDSI/SPKI. Porém, o modelo de autorização mantém camadas como suporte a alguns objetos CORBASec em nível de ORB (Figura 5.1): *SecurityManager*, *PrincipalAuthenticator*, *Credentials* e *Current*. São utilizados também os objetos do nível do ORB responsáveis por estabelecer comunicação segura: interceptadores de chamada segura, *Vault* e *SecurityContext*. Porém, os principais objetos do modelo que efetivam os controles de acesso estão concentrados no nível da aplicação, sugerindo então o uso das especificações CORBASec referentes ao nível 2 de segurança.

No nível SL2 do CORBASec aplicações cientes da segurança podem controlar as opções de segurança utilizadas nas invocações [69]: escolhendo a qualidade da proteção das mensagens, mudando os privilégios contidos nas suas próprias credenciais, escolhendo as credenciais que serão utilizadas na invocação de um objeto ou definindo quais credenciais poderão ser delegadas ou utilizadas no objeto destino. A flexibilidade oferecida pelo nível SL2 do CORBASec no manejo com credenciais é bastante adequada as necessidades de processamento com certificados SDSI/SPKI.

O modelo proposto para a integração do SDSI/SPKI com o CORBASec visa preservar a filosofia adotada no modelo SDSI/SPKI versão 2.0, onde um principal desejando obter acesso a algum recurso é inteiramente responsável pela busca das cadeias de certificados fornecendo-lhe o direito de acesso ao recurso. Ao servidor (guardião) cabe a função de verificações de autorização e de autenticidade do pedido.

A Figura 5.5 ilustra a disposição dos objetos presentes no lado cliente e servidor do protótipo – de acordo com o modelo estrutural do CORBASec (Figura 5.1). Como tecnologia de segurança foi adotado o SSL, devido à possibilidade de conversão dos certificados X.509 para o SDSI/SPKI .

Os interceptadores de chamada segura (*Secure Invocation Interceptor* – Figura 5.1) presentes no nível do ORB são utilizados no cliente e no servidor, possibilitando a distribuição de chaves, certificados, etc – além do estabelecimento de sessões seguras. Com as associações seguras entre cliente e servidor se garante as propriedades de integridade e de confidencialidade nas transferências das mensagens.

Os objetos *PrincipalAuthenticator*, *SecurityManager* e *Credentials* são parte do nível 1 do CORBASec, mas podem ser mantidos no SL2 porque contêm registros das informações de sessões do cliente e do servidor .

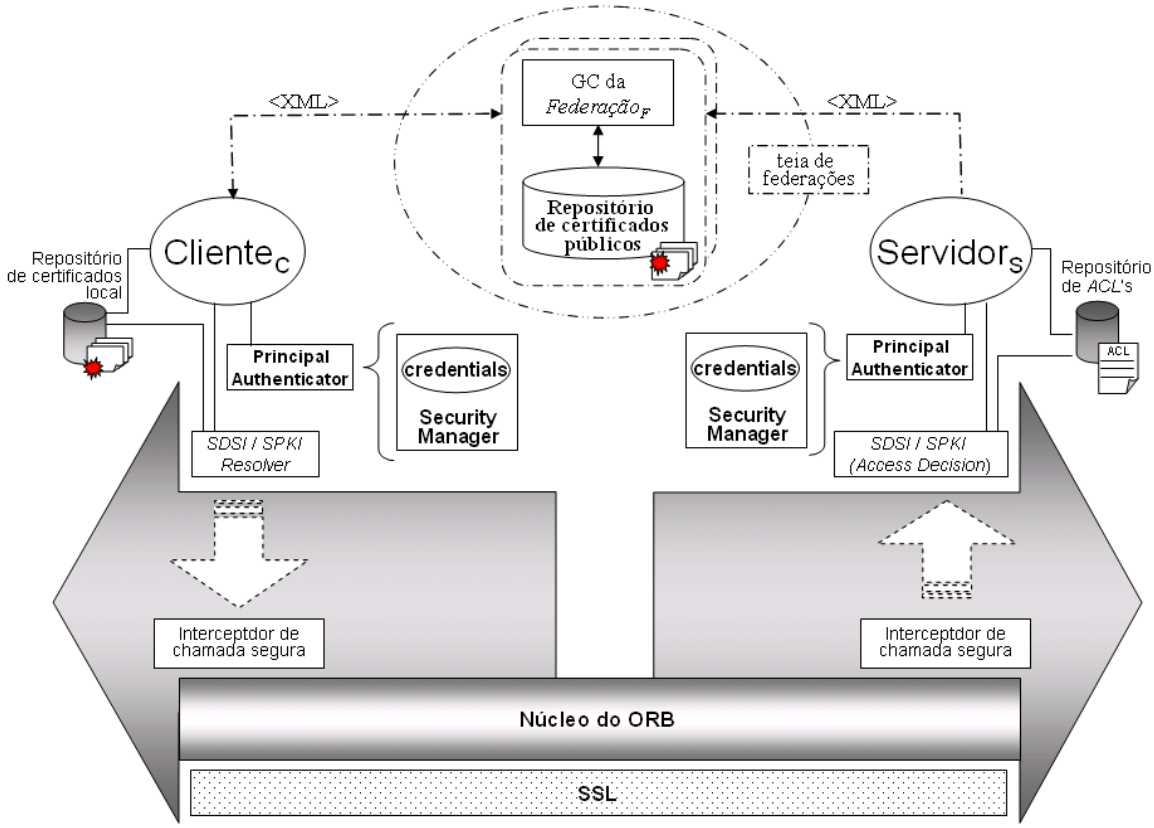


Figura 5.5 - Objetos do modelo de integração SDSI/SPKI - CORBASec

O objeto SDSI/SPKI *Resolver* permite a criação e a verificação de certificados (de nomes e de autorização), de seqüências (compondo as cadeias de certificados) e de assinaturas digitais. Além disto, provê os mecanismos necessários para o protocolo de autenticação mútua utilizado no modelo proposto. No objeto SDSI/SPKI *Resolver* – implementação parcial do agente (mostrado na Figura 4.2), são carregados os objetos SDSI/SPKI relativos à aplicação: par de chaves, certificados, seqüências e ACLs.

O objeto SDSI/SPKI *AccessDecision* provê mecanismos para o servidor verificar se uma requisição deverá ou não ser honrada. No modelo, o objeto SDSI/SPKI *AccessDecision* (guardião do serviço – mostrado na Figura 4.2) é acionado sempre que uma requisição for feita ao objeto servidor.

Basicamente, o objeto provê um método chamado "*access_allowed*" acionado a cada requisição para extrair os direitos fornecidos (pelo objeto cliente) das credenciais (contidas no objeto de sessão) e confrontá-las com os direitos requeridos lidos da ACL SPKI para decidir se permite ou não o acesso.

A. Autenticação mútua baseada na cadeia de autorização

O protocolo *challenge/response* implementado no protótipo segue as recomendações da especificação FIPS 196 - derivada da norma ISO 9798 parte 3 [70]. Neste caso, a autenticação é feita com base na infra-estrutura de chaves públicas, garantindo ao cliente e ao servidor a autenticidade das partes se comunicando, pois todas as requisições são assinadas digitalmente [71].

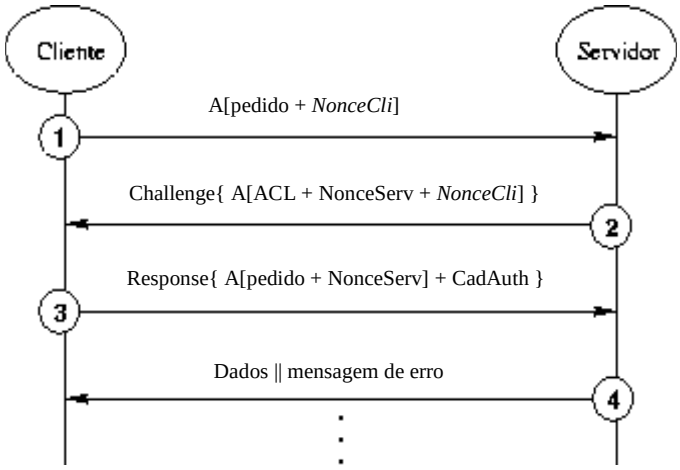


Figura 5.6 - Protocolo de autenticação *challenge*/*response* [68]

No cenário da Figura 5.6 um cliente deseja invocar um método em um objeto servidor. O método está protegido por uma ACL SDSI/SPKI e o guardião SDSI/SKPI executará o controle de acesso ao mesmo. A seguir será descrita a seqüência de trocas para que o cliente e o servidor se autentiquem mutuamente.

Na mensagem 1 o cliente faz uma requisição identificada por um *nonce* (mecanismo de proteção contra mensagens antigas), *NonceCli*, sem fornecer cadeia de autorização –

pois o cliente desconhece os direitos requeridos. O servidor aciona o guardião do serviço que gera um desafio (*challenge*) assinado por sua chave privada e o envia ao cliente. O desafio é composto pela ACL protegendo o recurso, pelo *NonceCli* e por um *nonce* do servidor, *NonceServ* (mensagem 2, Figura 5.6).

Utilizando a chave pública do servidor (contida na assinatura da mensagem) o cliente verifica a autenticidade do desafio. Em caso de sucesso, o cliente busca a cadeia de autorização que lhe dá o direito de acesso, compõe a resposta (*response*) e a envia ao servidor (mensagem 3). A resposta é formada pelo pedido original e pelo *NonceServ*, ambos assinados com a chave privada do cliente. Acompanha a mensagem também a cadeia de autorização (CadAuth, Figura 5.6).

O guardião SDSI/SPKI utiliza a chave pública do cliente (última chave da cadeia de autorização) para verifica a autenticidade da resposta (mensagem 3). Em caso de sucesso é verificada a cadeia de autorização. Se a seqüência da cadeia estiver correta a cadeia é autêntica, senão uma mensagem de erro é gerada (mensagem 4, Figura 5.6).

Note que no protocolo descrito acima a autenticação mútua está toda baseada sobre certificados de autorização SDSI/SPKI e, portanto, o que se tem na verdade é uma prova de autenticidade mútua das chaves públicas dos principais (cliente e servidor).

5.4.2 Considerações sobre o funcionamento do protótipo

Um objeto cliente ao ingressar no sistema de objetos CORBASec deverá autenticar-se através do objeto *PrincipalAuthenticator*. Este objeto cria um outro objeto, *Credentials*, contendo o conjunto de atributos de privilégios do cliente autenticando. Estes atributos de privilégio se encontram armazenados no objeto de sessão *SecurityManager* (Figura 5.5). No estabelecimento de uma sessão usando um canal seguro é necessária a autenticação mútua dos principais (cliente e servidor); foram implementadas funções para traduzir certificados de nome SDSI/SPKI (auto-assinados) em certificados X.509 utilizados pelo SSL, conforme o especificado em [42].

Depois de estabelecida a sessão entre o cliente e o servidor os atributos de autorização são transportados do lado cliente para o lado servidor a partir de certificados de autorização SDSI/SPKI – utilizados na forma de credenciais do contexto CORBASec. As ACLs SDSI/SPKI, neste caso, fazem o papel tanto do objeto *RequiredRights* quanto do

AccessPolicy do CORBASec: *RequiredRights* – pois definem os direitos requeridos para possibilitar a invocação de cada operação na interface; *AccessPolicy* – pois fornecem um conjunto de principais e uma lista de direitos requeridos para a execução de operações nos objetos de um domínio.

As cadeias de autorização buscadas pelo cliente para responder (*response*) aos desafios (*challenge*) lançados pelo servidor são transformadas em uma seqüência de bytes, transportadas do cliente para o servidor e adicionadas como atributos de privilégio nas credenciais definidas no servidor.

No servidor, o objeto SDSI/SPKI *AccessDecision* é automaticamente acionado a cada requisição de método. Os direitos do cliente (cadeia de certificados) são obtidos através do objeto SDSI/SPKI *ReceivedCredentials* – contido no objeto de sessão *SecurityManager* (Figura 5.5).

As ACLs SPKI (Figura 3.2) representam várias entradas, sendo cada entrada composta por uma chave pública e uma *tag*. A *tag* especifica o objeto e a operação que a chave pública pode efetuar.

5.4.3 Ferramentas

Na implementação do protótipo um conjunto de ferramentas foi utilizado para desempenhar diferentes funções. A seguir estas funções serão descritas junto aos respectivos elementos de software.

A. Middleware CORBA

O ORBacus v3.3.4 é um *ORB* (*Object Request Broker*) que está em conformidade com a especificação CORBA [72], mapeando completamente as IDL (*Interface Definition Language*) para as linguagens C++ e Java e tendo o IIOP (*Internet Inter-Orb Protocol*) como protocolo nativo. O ORBacus suporta os serviços básicos do CORBA (nomes, eventos, ...) além do suporte completo a programação dinâmica: invocação dinâmica de interfaces (DII), esqueleto de invocação dinâmica (DSI) e repositório de interfaces.

O adaptador de objetos BOA (*Basic Object Adapter*) é utilizado na versão 3.3.4 do ORBacus. O BOA é uma interface desenvolvida para ser amplamente disponível e para suportar uma ampla variedade de implementações de objetos comuns. Implementações de

objetos utilizando o BOA devem estar aptas a executar em qualquer ORB que suporte o mapeamento de linguagem necessário, embora o BOA seja geralmente dependente da implementação do ORB.

As especificações CORBA atuais [72] utilizam um adaptador de objetos portável (*POA - Portable Object Adapter*), independente da implementação do ORB.

O ORBacus é um ORB comercial, mas o seu uso em aplicações não comerciais é gratuito. No protótipo foi utilizado a versão 3.3.4 do ORBacus devido à restrição imposta pelo ORBAsec SL2 v2.1.6, pois o mesmo só foi projetado para a versão 3.3.4 do ORBacus.

O ORBAsec SL2 é um ORB seguro que fornece ao desenvolvedor meios para prover segurança em suas aplicações distribuídas na forma de autenticação e transferência de mensagens com criptografia forte. O ORBAsec SL2 está em conformidade com a especificação do nível 2 de segurança (Security Level 2) adotada na revisão 1.7 da especificação do serviço de segurança CORBA [69].

Como principais características o ORBAsec SL2 possui: funcionalidades do ORB (implementação em Java do ORBacus 3.3.x); funcionalidades do nível 2 de segurança; suporte ao protocolo SECIOP (*SECure Inter-Operability Protocol*); suporte ao Kerberos v5 (*GSS-API*), SSLv3 e comunicação insegura através do IIOP.

A implementação do ORBacus permite a introdução do ORBAsec SL2 através do ORBacus *Open Communications Interface* (OCI). Tal característica intrínseca do ORBacus permite acoplar mecanismos de transporte sob o GIOP (*General Inter-ORB Protocol*) propiciando o desenvolvimento da capacidade de autenticação e comunicação segura.

A implementação completa do *Security Level 2*, de acordo com a especificação CORBASec, é algo que não pode ser alcançado somente através do *ORB*. A solução completa envolve outros componentes em nível da aplicação que provêem serviços de gerenciamento como a administração de política e controle de acesso baseado nessas políticas. O ORBAsec SL2 não implementa os seguintes objetos:

- *Objeto Required Rights;*
- *Objeto Access Decision;*
- *Objeto Auditing Decision;*
- *Objeto Domain Access Policy*.

A especificação do *Security Level 2* não especifica como e nem onde os objetos citados acima deverão ser implementados e usados. Porém, esta também não impede o desenvolvedor de implementar as interfaces e utilizá-las em suas aplicações. O ORBAsec SL2 fornece as IDLs para estas interfaces.

B. Suporte ao SDSI/SPKI

Em [43] [42] foram definidas as características do SDSI/SPKI e a partir destes documentos surgiu uma implementação concreta do SDSI/SPKI. Desenvolvida em linguagem C, a biblioteca provê interfaces que permitem aos programadores utilizarem facilmente muitas das operações SDSI/SPKI . A distribuição ainda oferece um conjunto de ferramentas que auxiliam na criação, assinatura e emissão de certificados SDSI/SPKI. A implementação trabalha com pares de chaves RSA, porém não fornece rotinas para a criação das chaves.

Em [67] é implementada em linguagem Java uma biblioteca denominada JSDSI2.0. A biblioteca contém classes que provêem meios para converter e criar *S-expressions*, implementando os objetos fundamentais do SDSI/SPKI, certificados e chaves. A biblioteca possui uma ferramenta gráfica que permite criação de pares de chaves, gerenciamento de certificados, efetivação de assinaturas e verificação de cadeias de certificados.

O suporte criptográfico é conseguido através do *Cryptix3* – uma implementação das extensões de criptografia do Java (*JCE - Java Cryptography Extensions*). No protótipo foi utilizado o JSDSI2.0 por se tratar de uma implementação mais completa, já que existe a possibilidade de criação de pares de chaves sem a necessidade de utilizar outras ferramentas e principalmente por ter sido implementada em Java. Porém, como a primeira biblioteca citada acima, o JSDSI também trabalha somente com chaves RSA embora a especificação do SDSI/SPKI prevê o uso sistemas como o DSA (*Digital Signature Algorithm*).

C. Certificados SDSI/SPKI codificados em XML

O XML é um padrão da W3C (*World Wide Web Consortium*) cujo principal objetivo é o intercâmbio de conteúdos eletrônicos [73]. Baseado no SGML (*Standard Generalized Markup Language*), o XML possibilita a manipulação e visualização de dados para documentos com uma estrutura fácil de ser interpretada por humanos. O XML tem por

objetivo trazer flexibilidade e interoperabilidade a aplicações desenvolvidas sobre objetos web. Atualmente, o XML está sendo amplamente utilizado no ambiente Internet.

A biblioteca parserSxxS foi codificada por membros de nosso grupo [74] com base no DTD (*Document Type Definition*) especificado em [75] e pode ser utilizada como ferramenta para construir aplicações que trabalhem com o suporte SDSI/SPKI. A biblioteca é composta por dois pacotes: s2x e x2s. O pacote s2x provê as classes que codificam objetos SDSI (expressos em *S-expressions*) em documentos XML. O pacote x2s faz o inverso e provê classes que codificam documentos XML (objetos SDSI) em *S-expressions*.

O parserSxxS permitiu a adoção do formato de documentos XML como padrão unificado de armazenamento de objetos SDSI; no lugar de arquivos texto armazenado *S-expressions*. Assim, tanto aplicações que trabalham com *S-expressions* quanto com documentos XML podem ler e armazenar objetos SDSI/SPKI.

A implementação dos dois pacotes – s2x e x2s – contém classes com um mesmo esqueleto e seguem a mesma sistemática, porém com o corpo e funcionamento diferentes. A Figura 5.7 ilustra a classe principal s2xObject (das quais todas as outras classes derivam) e algumas das principais classes do pacote s2x.

Para cada objeto SDSI foi definida uma classe, a classe s2xParser é responsável por carregar um objeto SDSI expresso em *S-expressions*, determinar o tipo deste objeto e acionar a subclasse (do s2xObject) apropriada para que a mesma se encarregue de fazer a codificação em XML. Uma vez codificado em XML, o objeto SDSI é retornado como uma seqüência de caracteres, legível para humanos, para a classe s2xParser que o retorna para a aplicação. Desta forma, os objetos SDSI são separados em várias partes (chaves, certificados, etc) onde cada classe é responsável por codificar um tipo específico de objeto SDSI e a união do resultado de cada classe retorna o objeto recomposto em XML.

Por exemplo, um certificado de autorização é composto, basicamente, por duas chaves públicas e uma *tag*. Assim, a classe s2xParser invoca a classe s2xAuth, que por sua vez aciona as classes s2xPublicKey e s2xTag. O resultado produzido pela instanciação de s2xPublicKey e s2xTag é retornado para a classe s2xAuth que fornece o certificado de autorização expresso em XML.

O processo de conversão de objetos SDSI, expressos em *S-expressions*, consiste basicamente na manipulação de *strings* contidas na estrutura SexpList, implementada no

pacote SDSI da biblioteca JSDSI. Já a codificação de documentos XML em *S-expressions* é feita pelo *parser* XML presente no Java 2 (no caso, o DOM – *Document Object Model*) [76].

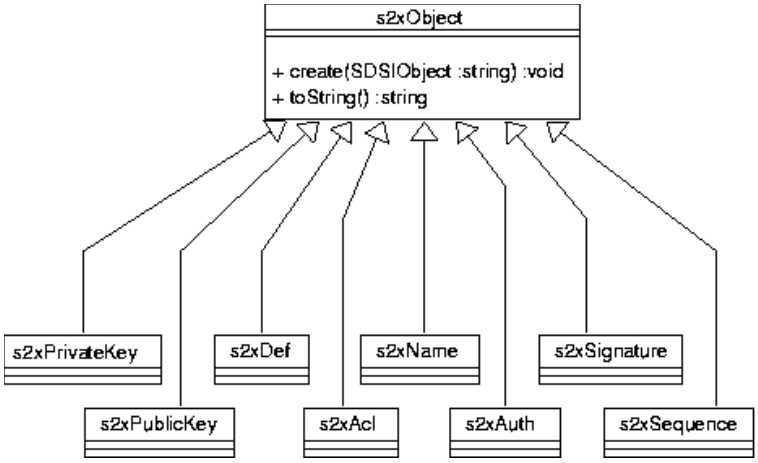


Figura 5.7 - Hierarquia parcial de classes da biblioteca parserSxxS [68]

D. Repositório de certificados

Objetos SDSI/SPKI são descritos em *S-expressions* [45], logo podem ser armazenados na forma ASCII (*American Standard Code for Information Interchange*). Por se tratar de uma estrutura simples, o seu armazenamento pode ser feito em um arquivo no formato texto, adotado na ferramenta presente na biblioteca JSDSI2.0.

Em [75] foi proposto uma forma padronizada de transformar objetos SDSI/SPKI codificados em *S-expressions* para documentos XML (*eXtensible Markup Language*) que serviu de referência para a implementação dos repositórios do protótipo.

No âmbito das teias de federações SDSI/SPKI há dois tipos de repositórios: os repositórios locais – próprios de cada membro da federação, e os repositórios globais públicos – localizados junto ao gerente da federação.

Implementados em Java, os repositórios são divididos em dois pacotes: *repository.local* e *repository.global*. O *repository.local* consiste em uma estrutura de arquivos representando os documentos XML armazenados no repositório local e um conjunto de operações que podem ser efetuadas sobre os documentos, tais como: inclusão, modificação, exclusão de arquivos ou busca de alguma informação dentro dos arquivos. A busca foi implementada com base no J2SE 1.4, através do *parser* SAX (*Simple API for*

XML) [77]. Os documentos XML armazenam as seqüências já formadas, o que acelera as buscas por este tipo de informação.

Para o repositório global foi utilizado o *Apache Xíndice* [78], um banco de dados que armazena documentos XML de modo nativo. O *Xíndice* utiliza o *XPath* [79] como linguagem para recuperação de documentos (*query language*). Logo, o pacote *repository.global* consiste basicamente na implementação de mecanismos eficientes para fazer consultas rápidas e simples, inclusões, exclusões e modificações na base de dados implementada no Xíndice.

Os certificados são armazenados em documentos XML, mas podem ser apresentados as aplicações neste formato ou em *S-expressions*, isto vai depender apenas da capacidade de manipulação dos certificados da aplicação. A conversão, se necessária, é feita utilizando o *parserSxxS* (seção 5.4.4-C).

5.5 Considerações sobre a aplicação de teste

O objetivo da aplicação de teste é avaliar a efetividade do modelo de autenticação e autorização proposto. Para efeito de simplificação da implementação não foram utilizados *applets* Java como previsto inicialmente. O uso de *applets* Java permitiria a utilização da aplicação em um *browser-http*. Porém, seria necessário considerar alguns aspectos de segurança com os quais não se desejava se preocupar nestes testes. Logo, a interface-cliente foi desenvolvida utilizando o pacote *Swing* do Java 2, com código estático, sem nenhum prejuízo para o resultado final do teste. Os resultados obtidos com esta aplicação de teste foram avaliados com relação ao seu desempenho na seção 6.5.

Os certificados X.509 traduzidos de certificados de nomes SDSI/SPKI foram armazenados através dos recursos oferecidos pelo Java para este fim (*Keystore*). Estes certificados são usados no estabelecimento de sessão segura entre as partes (cliente e servidor) através de uma sessão SSL.

O processo de inicialização das partes consiste na leitura do certificado X.509 (auto-assinados) e dos pares de chaves SDSI/SPKI. Os certificados X.509 são utilizados pelo SSL para estabelecer a sessão segura. As chaves SDSI/SPKI são utilizadas na efetivação e verificação de assinaturas.

Figura 5.8 - Tela de configuração da aplicação de teste [68]

Após o processo de inicialização cada parte estará apta a fazer ou receber invocações. A Figura 5.8 mostra a tela de configuração da aplicação de teste, que possibilita selecionar o par de chaves, o *Keystore*, o alias e entrar com as respectivas senhas.

5.6 Considerações sobre o modelo de implementação

Considerando as propostas apresentadas na literatura, é possível verificar que de alguma maneira as mesmas não seguem a filosofia do SDSI/SPKI. A preocupação com ataque por mensagens antigas, por exemplo, também não é considerada. Tanto Clarke quanto Lampinem [80] utilizam o envio de todas as credenciais do cliente para o servidor visando facilidades a implementação.

No trabalho de Clarke foi utilizado como aplicação um servidor http baseado em SDSI/SPKI, onde foram propostas algumas variações do protocolo de autenticação para diminuir a quantidade de trocas realizadas nesta fase e uma forma de restringir o acesso a ACL do recurso. Clarke trata o problema do envio da ACL através da imposição de uma outra ACL, cujo único objetivo é restringir o acesso à primeira. Tal abordagem é considerada devido a preocupação com a exposição dos principais que possuem direito no recurso. Porém, esta preocupação é inconveniente e desnecessária visto que seriam

demandadas diversas trocas de mensagens até o cliente conseguir obter acesso ao recurso, pois o mesmo terá de passar por no mínimo dois desafios (*challenges*).

O uso de duas ACLs deixaria obscuro para principais ainda não autenticados os direitos que cada principal contido na ACL possui, porém isto não se aplicaria aos principais com direitos sobre o objeto em questão.

No modelo o uso da ACL é feito no sentido de uma contextualização dentro da teia de federações SDSI/SPKI, pois se assim não fosse, poderia tornar-se possível durante o processo de negociação, a obtenção de uma cadeia falsa, por exemplo. Tal situação poderia ocorrer quando um cliente buscando uma determinada autorização, não conhecendo a chave pública do servidor de aplicação, negociasse a concessão de um direito de acesso a um servidor que não existe, por exemplo. Com o protocolo *challenge/response* o cliente antes de procurar a cadeia de autorização já conhece a chave pública do servidor de aplicação e quem detém o direito de acesso ao objeto que o cliente deseja. Este assunto é discutido com detalhes na seção 6.4.1.

O trabalho de Lampinem [80] é voltado para aplicações de objetos distribuídos CORBA e tem o seu funcionamento, com relação aos controles de autenticação e autorização, semelhantes à proposta de Clarke. Todos os certificados do cliente ficam disponibilizados em suas credenciais e assim são enviados para o servidor a cada requisição; o servidor deve selecionar as credenciais necessárias para cada acesso particular. Neste caso, assim como na proposta de Clarke pode-se lançar questionamentos com relação: ao sigilo, a escalabilidade e a não conformidades com a especificação atual do SDSI/SPKI .

Em geral nos trabalhos relacionados, observasse que um objeto-cliente disponibiliza todos os seus certificados de autorização em um único objeto *Credential* - enviado a cada invocação de um método. Este comportamento acarreta perdas de desempenho e não está em uníssono com a filosofia do SDSI/SPKI.

No modelo proposto, o cliente cria credenciais específicas para cada invocação e as envia uma única vez para a mesma requisição (na criação do contexto). Com isso são mantidas as premissas do SDSI/SPKI e não se expõe os demais certificados do cliente sem necessidade.

5.7 Considerações Finais

Neste capítulo foram abordados os principais tópicos referentes a integração do SDSI/SPKI ao ambiente do CORBASec; arquitetura e a implementação do protótipo e o modelo de integração. Foi considerado também o conjunto de ferramentas necessárias para a efetivação do protótipo.

6. Avaliação da proposta

6.1 Introdução

A avaliação de um sistema distribuído é sempre uma tarefa difícil, não interessando os aspectos que se considere. Nos dias atuais, estas dificuldades são ainda maiores; basta considerar a quase impossibilidade de garantir a correção de softwares complexos – na sua integração usando COTS (*Commercial Off-The-Shelf*) [63], por exemplo. Estas dificuldades são devidas as características abertas (*open*) destes sistemas e as dimensões das aplicações atuais, em grande parte, abrangendo a rede mundial.

No caso da segurança (*security*) as dificuldades são ainda maiores; as bases desta disciplina são técnicas fundamentadas na dificuldade computacional que ao contrário de sistemas confiáveis e tolerantes a falhas, não podem nem mesmo estabelecer uma previsibilidade probabilista de comportamento. O comprometimento da segurança de um sistema se dá em grande parte através de interações com o seu ambiente ou através da habilidade de manipulação dos recursos deste ambiente, exercida pelos intrusos [81]. Ou seja, as falhas dos mecanismos de segurança se devem a capacidade computacional dos atacantes e a dificuldade de prever o comportamento humano.

Porém, mesmo assim há esforços no sentido de avaliar o nível de segurança de sistemas ou produtos de segurança. São avaliações muito mais qualitativas do que quantitativas, envolvendo aspectos de construção e características do ambiente operacional destes sistemas. Estes métodos de avaliação não são definidos no sentido de garantir a

segurança destes sistemas; são antes de tudo metodologias de avaliação e testes. O padrão Critérios Comuns de Segurança (CC - *Commom Criteria*) [82] é uma destas iniciativas de metodologia de testes.

Neste capítulo será avaliada, segundo uma metodologia de testes, a segurança do protótipo desenvolvido, examinando desta maneira a adequação dos conceitos introduzidos nas extensões ao SDSI/SPKI. O protótipo é visto nesta avaliação como um produto de segurança. No método de avaliação usado é importante o conceito de *Gerência de Riscos* [83].

Em organizações normalmente são diferenciados os níveis de segurança em função das políticas de segurança adotadas no uso de suas informações. Estas políticas são resultantes de avaliações criteriosas feitas por especialistas, considerando sempre a relação custo / benefício, na definição de regras de segurança que objetivam levar uma organização a cumprir sua missão. Ou seja, as políticas assumem riscos quando consideram a freqüência e os danos de possíveis intrusões, e os custos para se prevenir de tais intrusões aos sistemas. Portanto, abordagens de avaliação concretas são aquelas baseadas na gerência de riscos.

São também considerados neste capítulo alguns testes de desempenho realizados no protótipo através de uma aplicação desenvolvida para ilustrar as potencialidades do modelo proposto.

A seguir, na seção 6.2 são brevemente abordadas metodologias de testes padronizadas – disponíveis para avaliação de segurança. Na seção 6.3 serão feitas considerações sobre vulnerabilidades. Na seção 6.4 será aplicada a gerência de risco ao protótipo. Na seção 6.5 se analisará aspectos relacionados ao desempenho do protótipo. Na seção 6.6 serão feitas considerações sobre o capítulo e na seção 6.7 se tecerão conclusões.

6.2 Metodologias de avaliação

Nesta seção serão mostrados alguns aspectos de dois exemplos de abordagens de testes de avaliação da segurança. O primeiro exemplo a ser considerado é o CC (*Critérios Comuns de Segurança*), que define metodologias padronizadas para uma agência credenciada gerar uma certificação classificando o nível de segurança do sistema. A segunda abordagem – usada em nossas avaliações – é fundamentada na *gerência de risco*

que ao contrário da anterior é baseada numa metodologia sistematizada executada por um especialista; o objetivo é atingir um nível aceitável de segurança no sistema.

O especialista, indispensável na segunda metodologia, geralmente define a política de segurança da instituição onde o sistema a ser avaliado será implantado, logo, conhece as características do ambiente onde o mesmo será utilizado. Assim, através da gerência de risco pode-se avaliar o *nível de susceptibilidade* as adversidades de uma organização.

Ambas as abordagens de avaliações citadas consideram o ambiente do sistema e possíveis cenários de vulnerabilidades e ameaças. No sentido do uso destas metodologias, na seção 6.4.2 item II, será apresentado um estudo sobre vulnerabilidades em sistemas computacionais. A análise do número de relatos de vulnerabilidades em sistemas reais permite montar as características do ambiente do sistema alvo da avaliação e, por conseqüência, definir as ameaças a que o mesmo estará sujeito.

6.2.1 Common Criteria

O *Common Criteria* (CC) é o resultado do esforço de várias organizações internacionais para desenvolver um único padrão de avaliação de segurança de sistemas de tecnologia da informação. O CC deriva de padrões anteriores: TCSEC (livro laranja de 1985), ITSEC (critério de avaliação europeu de 1991), CTCPEC (sistema de avaliação canadense de 1993) e do FC (união dos critérios canadense e europeu de 1993); tornou-se um padrão ISO (*Internacional Stardard Organization*) em 1995 – ISO/IEC 15408 [15].

Os critérios comuns estão fundamentados numa linguagem e em estruturas comuns para expressar requisitos de segurança de sistemas de tecnologia da informação. Estes fundamentos são utilizados no desenvolvimento de perfis de proteção (*PP - Protection Profiles*) e alvos de segurança (*ST – Security Targets*) que através dos requisitos especificam o comportamento do sistema. O *PP* define um conjunto de objetivos e requisitos, independente de sua implementação, para uma categoria de sistemas ou produtos de software. O *ST*, ao contrário, contém os requisitos de segurança de uma tecnologia ou um sistema e especifica as medidas funcionais e de garantia oferecidas pelo objeto da avaliação (*TOE – Target of Evaluation*) para atender a tais requisitos. O ST de um sistema pode incorporar requisitos ou declarar conformidade com um ou mais PPs. Os

STs e PPs refletem vários aspectos como as ameaças do ambiente, vulnerabilidades geradas no projeto, etc.

Como resultado da avaliação do ST (específico) para o sistema avaliado e os PPs próprios para a classe de aplicações do sistema, o *CC* gera um nível de garantia (*EAL - Evaluation* A*ssurance Level*) e classifica o *TOE* segundo uma escala progressiva de características de segurança sendo incorporadas. Na Tabela 6.1 é possível visualizar o equivalente dos EALs na classificação do TCSEC.

CC	**TCSEC**
-	D: Proteção mínima
EAL1	-
EAL2	C1:Proteção de segurança discricionária
EAL3	C2:Proteção de acesso controlada
EAL4	B1: Proteção de segurança com rótulos
EAL5	B2:Proteção estruturada
EAL6	B3:Domínios de segurança
EAL7	A1:Proteção verificada

Tabela 6.1 - Níveis de garantia dos Critérios Comuns em relação ao TCSEC

O CC considera que a segurança pode ser alcançada durante a fase de desenvolvimento, avaliação e operação do TOE. Durante a fase de desenvolvimento, através de refinamentos dos requisitos de segurança, gerando uma especificação sumária do TOE presente em um ST. Na operação do TOE vulnerabilidades podem surgir, então suposições preliminares sobre o ambiente operacional podem exigir revisões. Neste caso, o TOE deve ser modificado e a segurança reavaliada. A avaliação de um TOE é feita com base no respectivo ST e envolve a análise e testes do produto para verificar sua conformidade com o conjunto de requisitos de segurança.

A avaliação tem como principais entradas: o conjunto de evidências, o próprio TOE e o esquema de avaliação. O conjunto de evidências representa documentos de teste e projeto, códigos fonte e hardware. O esquema envolve a organização do processo como um todo, acordos entre entidades que executarão a avaliação e o órgão oficial que emitirá a classificação final. Está incluída no esquema também a adoção da metodologia de avaliação (*CEM – Commom Evaluation Methodology*), descrevendo as ações mínimas a serem executadas pelo avaliador na condução do processo. Todo este cuidado no processo

de avaliação se faz necessário para permitir múltiplas avaliações e reconhecimento internacional dos resultados fornecidos no uso do CC.

6.2.2 Gerência de risco

A gerencia de risco aplicada à segurança da informação teve sua primeira padronização reconhecida internacionalmente com a *British Standard*[20] 1799 (BS1799), publicação da parte 1 em 1995 e da parte 2 em 1998. Em 2000 tornou-se um padrão ISO[21], ISO/IEC 17799. Esta norma foi traduzida por órgãos oficiais de vários países, inclusive do Brasil. A ABNT[22] (Associação Brasileira de Normas Técnicas) traduziu tal padrão em 2001 como "NBR ISO/IEC 17799: Tecnologia da Informação – Código de prática para gestão de segurança da Informação". A ISO 17799 - parte 2 trata a gerência da informação em si, onde são feitas recomendações no sentido do entendimento do processo de avaliação do risco, porém não é definido um método para tal.

O NIST (*National Institute of Standards and Technology*) através de suas publicações da série 800 disponibiliza uma metodologia para a avaliação do risco na SP 800-30[23] (*Risk Management Guide for Information Technology Systems*) que será utilizada neste trabalho. Considerando as recomendações da SP 800-30, vulnerabilidades e possíveis ameaças ao sistema levam à definição de *níveis de exposição* à *adversidade*. O avaliador (especialista em segurança) considera então um conjunto de medidas de prevenção, adotadas para os casos identificados de vulnerabilidades e ameaças, no intuito de minimizar a exposição a tais adversidades.

As avaliações do sistema levam em consideração aspectos como a probabilidade associada às ameaças, capacidade de ataque do intruso, etc. Como tais probabilidades são atribuídas em função das características do ambiente e do histórico de incidentes, devido a complexidade do ambiente e o envolvimento de muitas variáveis de controle em muitos recursos, o resultado pode não ser determinista na prevenção de ameaças. Assim, o

[20] http://bsonline.techindex.co.uk/

[21] http://www.iso.ch/

[22] http://www.abnt.org.br/

[23] http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf

processo deve ser periodicamente repetido para que os especialistas verifiquem se o nível de exposição ao risco encontra-se em níveis aceitáveis – sem comprometimento do desempenho das funções da entidade sendo avaliada.

É assumido como risco o uso de uma vulnerabilidade do sistema, considerando a probabilidade de ocorrência de tal uso e o dano decorrente do mesmo. A *gerência de risco* corresponde a uma sistemática que dá consciência real do nível de suscetibilidade a vulnerabilidades no sistema sendo avaliado.

Nesta sistemática são definidas fases que envolvem o processo de identificação e de avaliação (*assessment*) dos riscos e a atenuação (*mitigation*) – adoção de medidas preventivas/corretivas que reduzem os riscos no sistema a níveis aceitáveis. Ou seja, a avaliação do risco envolve a identificação de vulnerabilidades e a avaliação do impacto do risco no sistema e, também, as recomendações para redução do risco. A atenuação do risco refere-se à priorização, implementação e manutenção das recomendações para redução do risco resultantes do processo de avaliação. É um processo cíclico continuado que abrange todos os níveis do sistema considerado (desde nível físico até de aplicação do modelo OSI).

6.2.3 Considerações sobre as metodologias

Por mais que validados por métodos formais e testes na fase de projeto, sistemas ou produtos de segurança durante suas utilizações geralmente apresentam falhas e vulnerabilidades. Os critérios considerados nestas validações são sempre baseados apenas no funcionamento esperado do sistema [84]. As situações anômalas criadas por intrusos em situações reais no sentido de provocar violações de segurança não são previsíveis.

Os testes padronizados do CC ou a sistemática da Gerência de Risco definem métodos padronizados para análises mais qualitativas que numéricas de sistemas ou produtos de segurança. O CC corresponde a uma metodologia que permite a emissão de certificação das garantias funcionais de segurança do sistema alvo da análise.

Políticas de segurança são independentes de implementações particulares. Sistemas ou mecanismos de segurança são implementações baseadas numa política de segurança. A metodologia de testes deve refletir o grau de coerência entre o modelo ideal (política) e o real (implementado). O método gerência de risco define um processo continuado que

permite através da análise de especialistas o aperfeiçoamento da segurança do objeto de análise referente às ameaças de seu ambiente, de tal modo que os cenários de comprometimento do sistema e os seus respectivos danos são considerados. Logo, as medidas para atenuar os seus efeitos também são levadas em conta.

A arquitetura proposta para o protótipo (seção 5.3) está baseada no uso de COTS. O CC se propõe a classificar o nível de segurança do protótipo. Adotamos neste trabalho a gerência de risco para avaliar o protótipo como um todo e o CC para validar partes funcionais do mesmo.

6.3 Vulnerabilidades

As vulnerabilidades, conforme citado no capítulo 2 deste texto, são em geral a origem de situações adversas provocadas ou descobertas acidentalmente em sistemas computacionais. É de conhecimento público que vulnerabilidades são muito freqüentes e não se pretende esgotar o assunto neste tópico. Mas para efeito de análise da segurança de sistemas é importante certa quantificação dos tipos mais prováveis de vulnerabilidades encontradas em sistemas reais.

Assim, na próxima seção são considerados os casos identificados como mais comuns, onde a exploração bem sucedida destas vulnerabilidades, através de ataques, resulta no comprometimento ou violações do sistema. Analisando o número de relatos de vulnerabilidades publicados em periódicos, *papers* e alertas de segurança emitidos por órgãos especializados no assunto (CERT/CC Advisories, *SecurityFocus*, Bugtrap, por exemplo), é percebido de imediato que mesmo sistemas implementados segundo as recomendações do CC sobre desenvolvimento de *TOE* ou de qualquer outra metodologia de concepção, freqüentemente apresentam vulnerabilidades [85]. Tal fato é resultante do número significativo de complexas combinações dos elementos que compõem um sistema executando num ambiente computacional de dimensões consideráveis e ainda, de erros das mais diversas naturezas oriundos do processo de desenvolvimento e/ou de implantação.

Nas análises ciatadas acima e mesmo na publicação dos alertas de segurança há dificuldades básicas com relação ao número considerável de definições, interpretações e usos do termo *vulnerabilidade*. Para unificar a utilização do termo em 1999 foi criada no

MITRE Corporation[24] uma lista de nomes padronizados para vulnerabilidades (*vulnerabilities*) e outras informações de exposições (*exposures*) de segurança que foi denominada *Common Vulnerabilities and Exposures* (CVE) [85].

O CVE é um dicionário de vulnerabilidades que atualmente conta com aproximadamente 2573 registros no *CVE list* propriamente dito e 3300 registros na lista de *CVE candidate* – vulnerabilidades e exposições relatadas ao *Editorial Board* que se encontram em processo de avaliação, se aceitas, passarão a compor a *CVE list* [86].

O NIST mantém uma base de dados de vulnerabilidades padronizadas, com refinamentos (*fine-grained*), que liga as vulnerabilidades publicamente conhecidas as informações de correção (*patch/service pack,...*) das mesmas. Esta base de vulnerabilidades é conhecida como ICAT ou *"ICAT Metabase Documentation - CVE Vulnerability Search Engine"*[25]; O ICAT é baseado no dicionário CVE.

A seguir serão descritos brevemente os tipos de vulnerabilidades que se encontraram relacionadas no ICAT.

6.3.1 Tipos de vulnerabilidades

Os tipos de vulnerabilidade mais comuns que foram catalogados e padronizados para constar na base de vulnerabilidades do ICAT serão descritos a seguir:

- **Erro de validação da entrada de dados** (*input validation error*): vulnerabilidade caracterizada pela entrada de dados não propriamente verificada, propiciando uma seqüência de entradas maliciosas ou mal formadas. Este tipo de vulnerabilidade pode apresentar-se em duas variações:
  - **Extrapolação de limite** (*boundary overflow*): ocorre quando um limite assumido pelo sistema é excedido e causa um mau funcionamento. Estão nesta classe: problemas na extrapolação dos limites de memória, disco e largura de banda de comunicação, por exemplo.

[24] http://www.mitre.org

[25] http://icat.nist.gov/

  - **Extrapolação de limites de uma memória temporária** (*buffer overflow*): uma entrada de dados que excede um limite de tamanho esperado caracteriza esta vulnerabilidade e provoca este erro de validação. Neste sentido, o atacante (intruso) pode colocar um conteúdo (código) de seu interesse na parte que ultrapassa o tamanho do buffer para tirar proveito da situação subseqüentemente.
- **Erro de validação de acesso** (*access validation error*): é caracterizado por mau funcionamento no mecanismo de controle de acesso.
- **Erro na manipulação de uma exceção** (*exceptional condition handling error*): é a introdução de um erro devido à ocorrência de uma condição de exceção para o qual o manipulador definido (ou a sua ausência) não prevê tratamento adequado.
- **Erro devido ao ambiente** (*environmental error*): é a situação em que o sistema é vulnerável ao ambiente onde o mesmo está instalado. Tal caso pode ocorrer como resultado de uma interação não prevista entre ambos (sistema e ambiente), por exemplo.
- **Erro de configuração** (*configuration error*): ocorre quando os parâmetros de configuração de um sistema são manuseados pelo usuário de maneira errônea, tornando o sistema vulnerável.
- ***Race condition***: caracterizado por verificações não-atômicas de segurança que levam a geração de inconsistências. Por exemplo, num sistema quando uma ACL é ativada para a verificação das permissões de uma operação, entre a verificação e a efetivação da operação pode ocorrer uma alteração da ACL. Esta modificação pode ser tal que a operação antes permitida pela ACL, após a alteração torna-se inválida. Assim, se a consulta foi feita antes da modificação da ACL, com a execução da operação é produzida uma violação da política de segurança do sistema. Atacantes podem explorar situações similares a esta para tirar proveito de curtos períodos de inconsistência ou de não sincronização.
- **Erro de projeto** (*design error*): quando o sistema apresenta erros de implementação ou de configuração que são originários de falhas de projeto se tem vulnerabilidades por erro de projeto.

6.3.2 Ferramentas baseadas em CVE

Na URL *http://cve.mitre.org/compatible/product_type.html* há uma relação de produtos e serviços classificados por categoria: *Vulnerability Database*, *Security Advisories and Archives*, *Intrusion Detection and Management*, *Intrusion Monitoring and Response Service*, etc. Nesta relação, é encontrada uma vasta gama de ferramentas declaradas CVE-compatíveis para auxiliar no processo de gerência de risco na detecção de vulnerabilidades. Apenas a título de ilustração, constatou-se que boa parte das ferramentas se autoproclamam capazes de identificar e gerar relatório de aproximadamente 20.000 incidentes de segurança que têm origem em vulnerabilidades.

O *MITRE Corporation* mantém também, um grupo para mediar a padronização de uma linguagem de avaliação de vulnerabilidades em domínio público (*Open Vulnerability Assessment Language – OVAL*[26]). Esta linguagem utiliza *SQL* (*Structured Query Language*) para escrever *queries* representando registros do dicionário CVE. Ou seja, um grupo de especialistas discute como codificar registros de vulnerabilidades do dicionário CVE em *queries SQL*; o objetivo das *queries SQL* é obter informações de configuração e características de um sistema – incluindo versão do sistema operacional da máquina e das aplicações instaladas. Usando as *queries SQL* em um sistema ou aplicação é possível concluir se o mesmo apresenta as vulnerabilidades presentes no dicionário CVE. Enquanto as *queries* vão sendo escritas, uma para cada entrada do *CVE list*, vão sendo agrupas em esquemas.

Os esquemas contêm todas as *queries* que se referem a uma plataforma, por exemplo, Windows XP. Atualmente, *OVAL* tem esquemas padronizados para Windows (NT4.0, 2000 e XP), Linux (Redhat e Debian) e Solaris (7 e 8). Como resultado de uma avaliação através de um esquema é produzido um relatório com as vulnerabilidades encontradas no sistema avaliado.

[26] http://oval.mitre.org/

Além do SQL, OVAL suporta a escrita de *queries* em XML como um formato alternativo para aplicações que desejarem produzir documentos XML como relatório da avaliação por um determinado esquema.

6.3.3 Tratamento de vulnerabilidades

Basicamente, para tratar ou controlar vulnerabilidades algumas técnicas podem ser identificadas da literatura [87]: técnicas de prevenção (*avoidance*), de eliminação (*elimination*) e de tolerância (*tolerance*).

A prevenção inclui técnicas de projeto e/ou de instalação visando evitar um tipo particular de vulnerabilidade.

Técnicas de eliminação são caracterizadas pela detecção e identificação de vulnerabilidades subseqüentemente corrigidas por ações visando a eliminação das mesmas.

Técnicas de tolerância admitem a existência de vulnerabilidades no sistema ou nos produtos de segurança e através do uso de redundâncias tentam limitar os impactos associados às mesmas.

6.4 Gerência de riscos aplicada ao protótipo

A susceptibilidade a adversidades do protótipo desenvolvido neste trabalho é avaliada neste item segundo registros de incidentes de segurança de sistemas reais. Os controles propostos pela metodologia gerência de riscos no sentido de atenuá-los ou neutralizá-los são considerados ainda na fase de desenvolvimento do protótipo. Por fim, se considera a hipótese de comprometimento do sistema, avaliando-se os riscos e os possíveis danos causados nesta situação. Então, considerando esta situação se avalia se os riscos estariam em níveis aceitáveis sem que as devidas correções das vulnerabilidades fossem efetivadas [88].

A fase de desenvolvimento do software é apenas um subconjunto dos aspectos a se considerar num processo de avaliação da gerência de risco como um todo. Como comentado na seção 5.4, o protótipo apresentado naquela seção é apenas uma parte de um projeto maior, onde as federações estão efetivamente sendo implementadas. Quando a

aplicação e o ambiente como um todo estiverem definidos será possível utilizar o CC para validar a aplicação e o suporte de autenticação e de autorização desenvolvidos para o ambiente distribuído pretendido.

O uso da gerência de risco se fez no sentido de identificar possíveis vulnerabilidades presentes no sistema. Então num processo de realimentação continua, buscam-se soluções a serem incorporadas no sistema de modo a minimizar possíveis riscos de segurança.

Consideramos para efeito de avaliação das características do suporte de autenticação e autorização proposto, um cenário como o mostrado na seção 4.6 e avaliamos os riscos mais comuns para tal cenário com base na Tabela 6.2. Esta consideração foi feita porque o cenário proposto para exemplificar o esquema reflete uma possível aplicação de comércio eletrônico.

Uma metodologia como o CC, basicamente, limita os testes de identificação de vulnerabilidades àquelas provenientes de erros de programação. A gerência de risco parece mais adequada à avaliação da arquitetura do protótipo, pois considera todos os tipos de vulnerabilidades apresentadas na seção 6.3.1 [89]. Ou seja, considera também possíveis erros provenientes da interação entre os componentes (COTS) do protótipo dentro do ambiente distribuído. Nesta fase do projeto este tipo de avaliação não poderá ser aplicado por completo.

A gerência de risco pode não parecer adequada como uma avaliação final de um software ou protótipo porque depende da experiência de um especialista para ter resultados mais precisos, mas quando aplicada ao ciclo de desenvolvimento do protótipo como foi o caso neste trabalho, permitiu utilizar uma sistemática que influenciou positivamente as características finais do protótipo. Neste caso, potencialidades e/ou deficiências que seriam percebidas quando da avaliação final do protótipo puderam ser consideradas ainda durante a implementação do mesmo.

Na seqüência serão avaliadas possíveis vulnerabilidades do protótipo considerando registros comuns de incidentes de segurança.

6.4.1 Avaliação de riscos

É o primeiro passo da sistemática da gerência de risco. A avaliação de riscos determina a abrangência de ameaças em potencial e o impacto (magnitude do dano) que

podem causar ao sistema. Este processo é composto por 9 fases: caracterização do sistema, identificação de ameaças, identificação das vulnerabilidades, análise de controle, definição de probabilidades, análise de impacto, determinação dos riscos, recomendações de controle e documentação dos resultados.

No caso da avaliação de riscos aplicada ao protótipo, incidentes que podem ser resultantes das características da aplicação, devido a semântica da mesma, por exemplo, não forão considerados neste trabalho. Assim, os aspectos considerados são principalmente os ligados ao projeto e a implementação do suporte de autenticação e de autorização da arquitetura proposta.

I. Caracterização do sistema

Como o nome do tópico dá a entender nesta seção são descritas as informações sobre o sistema implementado. A seguir serão listadas algumas das características do esquema:

a) PKI baseada em cadeias de confiança;
b) Modelo de confiança igualitário;
c) Identificação de principais baseada em chaves públicas, ao invés de ID;
d) Forte vínculo entre a identificação (chave pública) e a prova de autenticidade – através da assinatura digital;
e) Nomes locais, mas de alcance global – através da vinculação dos nomes locais as chaves públicas. Logo, a política de autenticação é totalmente descentralizada;
f) Modelo baseado na autorização e no cliente, ao invés de ser focado na autenticação e no servidor;
g) Autorização codificada em certificados sendo, portanto a política de autorização totalmente distribuída e descentralizada;
h) Sistema de gerência de usuários descentralizado (sem necessidade de gerência de contas/*accounts* e senhas);
i) Modelo de confiança independente de tecnologia e
j) Protótipo integrado aos sistemas vigentes através do *middleware* CORBA.

II. Identificação de vulnerabilidades

Nesta etapa se busca observar e relacionar as possíveis vulnerabilidades ou falhas que podem ser exploradas no sistema. Tomando como base a seção 6.3.1, pode-se considerar todos os tipos de vulnerabilidades candidatos diretos ou indiretos a comprometer o sistema, quando explorados com sucesso.

A Tabela 6.2 apresenta um levantamento feito, tomando como base os registros de vulnerabilidades do ICAT. Os tipos e os objetos de vulnerabilidades – que podem ter alguma relação com o protótipo desenvolvido neste trabalho – serão relacionados nesta tabela com as respectivas ameaças, ataques e violações. Os itens da Tabela 6.2 estão ordenados por freqüência de ocorrências no registro de alerta do CVE *list*, compreendendo o período de 1999 a 2002.

Na seção que se segue, no subitem IV, os tipos de vulnerabilidades indicados na Tabela 6.2 são examinados considerando a susceptibilidade do protótipo aos mesmos. No subitem VIII recomendações adicionais serão feitas para evitar um possível comprometimento do sistema com base no levantamento da Tabela 6.2.

Tipos de vulnerabilidade	**Objeto (de ocorrência) das vulnerabilidades**	**Violação, ataque ou ameaça**	**Ocorrências**	**percentual**
Qualquer uma ou combinação das citadas na seção 6.3.1	vários	***Denial of service***	596	23,0%
Erros de validação de entrada de dados	***Buffer overflow***	vários	541	21,0%
Erro de configuração, erro devido ao ambiente, erro de projeto	***Password, Authentication***	***Bypass controls***	135	5,0%
Erros de validação de entrada de dados	***Format string***	vários	67	3,0%
Erro devido ao ambiente	vários	***Vírus, Trojan/ Malicious code***	46	1,5%
Erro de projeto	vários	***Memory leak***	40	1,5%
Erro devido ao ambiente	vários	***Spoofing***	35	1,0%
Erro de projeto	vários	***TCP prediction***	32	1,0%
Erro de projeto, erro de configuração	***Java***	vários	28	1,0%
Race condition	Ambiente com compartilhamento de recursos	vários	25	1,0%
Erro de projeto, erro devido ao ambiente	vários	***Hijacking***	16	0,5%
Erro de projeto, erro de configuração	vários	***Insertion / Injection*** *de conteúdos*	8	0,0%
Erro de projeto	***Challenge-response***	*Bypass controls*	5	0,0%
Erro de projeto	vários	***Replay***	2	0,0%
Erro de projeto, erro devido ao ambiente	vários	***Man-in-the-middle***	1	0,0%

Tabela 6.2 – Incidentes mais comuns nos boletins de alertas de segurança

III. Identificação de ameaças

O objetivo desta etapa é identificar as ameaças que podem explorar vulnerabilidades no sistema. Uma ameaça é caracterizada como uma entidade com potencial para explorar com sucesso uma vulnerabilidade. Assim, pode-se citar como exemplo de ameaças, principalmente, softwares maliciosos (vírus, *trojan horse* / *malicious code*) e *hackers* (intrusos internos ou externos a rede).

IV. Análise de controle

Esta fase tem como objetivo a identificação dos controles necessários para minimizar ou eliminar as possibilidades de exercício de uma ou mais vulnerabilidade resultando no comprometimento do sistema. Os controles necessários para atenuar ou neutralizar os efeitos negativos do comprometimento do sistema são conseqüência destas considerações. Com base na Tabela 6.2 são feitas considerações a respeito da susceptibilidade do protótipo a alguns tipos de vulnerabilidades, ataques e violações:

a) **Negação de serviço** (*denial of service*): esta violação de segurança é bastante difícil de ser evitada, uma vez que, pode ser fruto de ataques via rede, do consumo excessivo de recursos da máquina ou ainda, pode ser resultante da exploração de qualquer tipo de vulnerabilidade que implique na indisponibilidade do serviço ou de um recurso. Não faz parte das premissas do suporte de autorização e autenticação proposto aplicar técnicas para evitar a negação de serviço. Porém, para minimizar seus efeitos um principal pode publicar seus certificados públicos em mais que uma federação e, a federação pode replicar seu repositório. Assim, se ganha um grau considerável de tolerância a este tipo de ataque.

b) ***Buffer Overflow e Format String***: estas vulnerabilidades (sub-conjunto de erros de validação de entrada de dados) são características de programas escritos em linguagem “C”. Como na implementação do protótipo foi utilizada a linguagem Java se presume que o protótipo esteja imune as mesmas.

c) ***Bypass controls***: tipo de ataque ligado aos serviços de autenticação e de autorização em que se evita as regras vigentes de algum tipo de política. Como no

protótipo a autenticação é efetuada por assinatura digital e as políticas de autorização estão baseadas em chaves públicas que provam seus direitos fazendo uso de cadeias de certificados de autorização, parece pouco provável que o *bypass control* ocorra. Porém, se um *bypass* ocorrer por algum motivo seu dano estará limitado a uma requisição específica, pois no protótipo a cada nova requisição (de operação) os certificados de autorização devem ser apresentados – para comprovar os direitos necessários. Vulnerabilidades que tornem este ataque mais efetivo são mais susceptíveis em sistemas clássicos, pois uma vez autenticado o principal pode executar um conjunto de ferramentas e ações que lhe permitam ganhar o controle do sistema e/ou prepará-lo para acessos subseqüentes, através da instalação de *backdoors*, por exemplo.

d) ***Memory leak***: vulnerabilidades que permitam *canais de memória* são facilmente usadas tanto por códigos escritos em linguagem "C" [90] quanto em Java [91]. Assim, no protótipo foi usada uma ferramenta que identifica *memory leaks* em códigos Java, o *JProbe Memory Debugger* [92]. Ou seja, este tipo de ameaça foi minimizado em tempo de desenvolvimento.

e) ***TCP prediction, Spoofing e Hijacking***: esta classe de ataque não afeta o nosso protótipo, dado que a troca de mensagens está sendo feita em sessão cifrada (canais de comunicação seguros). O estabelecimento destes canais envolve a troca de certificados, ou seja, envolvem a autenticação mútua (*challenge-response*). Todas as medidas clássicas foram tomadas no processo de autenticação mútua para evitar o *hijacking*. Supondo que um ataque deste possa ser bem sucedido, mesmo assim o dano não seria significativo, pois em qualquer requisição feita por um cliente é exigida a assinatura digital do mesmo (efetuada com a chave privada do cliente)[27]. Para forjar uma requisição é necessário também comprometer a chave privada do cliente.

f) ***Race condition***: a detecção de *race condition* em aplicações *multi-threads*

[27] Quando a requisição chega ao servidor é verificada a veracidade da assinatura da requisição utilizando-se a chave pública do último sujeito da cadeia de autorização apresentada para comprovar o direito de acesso (maiores detalhes na seção 5.4.3).

baseadas em Java [93], por exemplo, pode ser efetuada com um software como o JProbe Threadalyzer [94].

g) ***Man-in-the-middle***: um dos ataques mais danoso a sistemas que usam controles criptográficos é o *man-in-the-middle*, principalmente em esquemas que envolvem trocas de chaves. No caso do protótipo, nas trocas de autenticação do protocolo de autenticação *challenge-response* as mensagens são assinadas digitalmente por chaves assimétricas conhecidas a priori através de cadeias de autorização [95]. Não parece existir evidências que propiciem este tipo de ataque no nosso protótipo.

h) ***Replay***: para evitar esta vulnerabilidade na arquitetura do protótipo em todas as trocas de mensagens são usadas técnicas de *nonce* (*timestamp*). Considera-se praticamente nulo o risco deste ataque se tornar bem sucedido no protótipo.

i) **Ataque por inserção de conteúdo/código**: como relatado em [96] na troca de mensagens pode haver inserção de conteúdos nas mensagens em trânsito, forjado através da sobreposição de pacotes/segmentos de uma mensagem. Com o uso de canais seguros (cifragem de mensagens) este tipo de ataque parece pouco provável. Porém, mesmo considerando o sucesso na quebra da cifragem da comunicação, a assinatura digital das solicitações impediria o sucesso nestes tipos de ataque porque se detectaria a violação da integridade da mensagem. Ataques deste tipo não parecem aplicáveis para o caso do protótipo.

j) **Vírus, *trojan horse* / *malicious code***: a gama de abrangência deste tipo de ameaça, identificada classicamente como o "problema de programas emprestados", não se aplica exatamente à implementação do sistema de autenticação e autorização proposto.

A maior preocupação com este tipo de ameaça, para o caso do protótipo, é com relação ao comprometimento da chave privada do principal. Programas como *spies*[28] ou *keystroke loggers*[29] podem registrar informações de segurança da

[28] http://www.spy-software-planet.com/spyshadow.htm

[29] http://www.ntsecurity.nu/toolbox/klogger/

máquina local e, conseqüentemente, do protótipo. Os controles impostos pelos mecanismos de canal seguro do protótipo limitam possíveis envios destas informações a receptores conhecidos.

k) ***Fake authorization chain***: um membro da federação pode ficar tentado a inserir no GC de sua federação uma cadeia de autorização falsa, ou melhor, a cadeia poderia ser válida, mas a primeira chave da cadeia (chave do servidor) poderia ser falsa. O objetivo do principal que fizesse isto seria tirar algum proveito da situação durante a fase de negociação do processo de criação de novas cadeias. Ou seja, colocando a cadeia falsa na federação o principal (intruso) poderia tentar enganar um outro principal (cliente) que efetuaria uma busca na federação tentando identificar um direito de acesso a um servidor. O cliente identificaria a cadeia (falsa) e negociaria com o intruso a concessão do direito de acesso e só descobriria que a cadeia é falsa quando tentasse acessar o servidor (seria o equivalente a "cair no conto do vigário", por exemplo). O protótipo foi implementado usando o protocolo *challenge-response*, onde o cliente inicialmente envia uma mensagem ao servidor e solicita sem permissão (sem a cadeia de autorização) a efetivação de uma operação. O servidor informa, através da *ACL-SPKI* que protege o objeto o principal (chave pública) que tem acesso ao objeto e o direito de acesso necessário. Então, o cliente busca a cadeia de autorização com tal direito de acesso. Observa-se que o cliente não possuindo a cadeia de autorização necessária fará a busca na federação indicando a chave do servidor e o direito desejado. Logo, se um certificado falso for colocado na federação, como descrito acima, o mesmo nem será selecionado pelo algoritmo de busca porque os critérios de busca indicados pelo cliente (servidor e direito de acesso) não serão satisfeitos. Portanto, não se consegue identificar uma situação onde uma cadeia de autorização falsa pudesse ser inserida na federação e o esquema proposto pudesse ser corrompido.

l) **Ponto único de falhas e vulnerabilidades:** o GC, ligando o cliente ao servidor, não se caracteriza como um ponto centralizador no papel de mecanismo de liberação de cadeias de confiança, pois sua atuação limita-se a gerência das filiações e aos mecanismos de busca das cadeias de autorização e de nomes. O

comprometimento ou falha de um GC não significa o isolamento de um principal. Servidores e clientes podem publicar seus certificados em vários GCs para evitar possíveis dificuldades com a indisponibilidade de federações.

m) **Comprometimento da chave privada:** este tipo de situação é muito difícil de ser detectada. Uma estratégia para minimizar o efeito do comprometimento pode ser vista no item VIII, desta seção. Uma vez detectado o comprometimento de sua chave privada, o usuário deve emitir um certificado de revogação da referida chave e publicá-lo, inclusive no GC da sua federação.

n) **Challenge-Response**: o protocolo de autenticação *challenge-response* implementado no protótipo segue as recomendações da especificação FIPS 196, como comentado na seção 5.4.3. As trocas de autenticação são feitas com base no sistema criptográfico de chave pública, logo não se vê como este tipo de protocolo pode se tornar vulnerável no protótipo.

V. Determinação de probabilidades associadas a vulnerabilidades

O objetivo desta atividade é compor taxas de probabilidade de uma vulnerabilidade ser explorada com sucesso por uma ameaça, resultando possivelmente em uma violação ou ataque.

Ataque ou violação	probabilidade
Negação de serviço (*denial of service*)	média
Buffer Overflow e Format String	baixa
Bypass controls	baixa
Memory leak	baixa
Race condition	baixa
TCP prediction, Spoofing e Hijacking	baixa
Man-in-the-middle	baixa
Replay	baixa
Fake authorization chain	baixa
Ponto único de falhas e vulnerabilidades	baixa
Ataque de inserção de conteúdo / código	baixa
Comprometimento da chave privada	baixa

Tabela 6.3 - Probabilidade de uma vulnerabilidade ser explorada com sucesso

As probabilidades da Tabela 6.3 são definidas segundo os seguintes critérios [83]:

o) *alta*: se a ameaça está altamente motivada e suficientemente capacitada para explorar a vulnerabilidade e os controles de prevenção são ineficientes.

p) *média*: se a ameaça esta motivada e suficientemente capacitada para explorar a vulnerabilidade e os controles de prevenção podem impedir o ataque ou violação.

q) *baixa*: se a ameaça está pouco motivada ou se a capacitação não é suficiente para explorar a vulnerabilidade, ou ainda, se os controles de prevenção são suficientes para inviabilizar ataques baseados nestas vulnerabilidades.

As atribuições de probabilidades da Tabela 6.3 foram efetuadas considerando-se o cenário da Figura 4.11 e a análise de controle do protótipo considerada na seção 6.4.1, item IV. O objetivo principal é ensaiar o comportamento do protótipo dentro do ambiente do cenário, aplicando-se a metodologia.

VI. Análise de impacto

Esta fase visa avaliar o impacto que o resultado adverso de um ataque ou violação bem sucedida causa à instituição. Como o protótipo não é parte do conjunto de softwares de uma corporação foram atribuídos níveis de impacto comuns (a cada item da Tabela 6.4) para uma aplicação como a do cenário da Figura 4.10.

Ataque ou violação	**Impacto**
Negação de serviço (*denial of service*)	médio
Buffer Overflow e Format String	baixo
Bypass controls	médio
Memory leak	médio
Race condition	baixo
TCP prediction, Spoofing e Hijacking	baixo
Man-in-the-middle	médio
Replay	baixo
Fake authorization chain	baixo
Ponto único de falhas e vulnerabilidades	baixo
Ataque de inserção de conteúdo / código	baixo
Comprometimento da chave privada	médio

Tabela 6.4 - Magnitude do impacto adverso causado pelo exercício de uma vulnerabilidade

Na Tabela 6.4, o nível de impacto foi definido de acordo com os seguintes critérios [83]:

- *alto*: se o uso da vulnerabilidade pode provocar custosas perdas de um recurso ou afeta de maneira definitiva a missão, reputação ou o interesse da organização.
- *médio*: se o exercício da vulnerabilidade pode resultar perdas ou afetar a missão, reputação ou ainda o interesse da organização.
- *baixo*: se o uso da vulnerabilidade resultar em perdas não muito significativas ou ainda afetar levemente a missão, reputação ou o interesse da organização.

VII. Determinação do nível de risco

O objetivo desta fase é determinar o grau de susceptibilidade ao risco que cada vulnerabilidade representa considerando a probabilidade da mesma ser explorada com sucesso e a magnitude do impacto adverso que o fato causaria no sistema. O produto destes dois fatores resulta o grau de risco de cada ataque ou violação.

Objetivando-se chegar a um valor numérico para o nível de risco, foram assumidos os seguintes valores para as probabilidades da Tabela 6.3: alta = 1, média = 0,5 e baixa = 0,1. Os valores de impacto da Tabela 6.4 foram convertidos em números da seguinte maneira: alto = 100, médio = 50 e baixo = 10. A Tabela 6.5 mostra o nível de risco resultante do produto dos dois fatores (probabilidade x impacto) considerados acima.

Ataque ou Violação	**Probabilidade**	**Impacto**	**Nível de Risco**	
Negação de serviço *(denial of service)*	0,5	50	25	médio
Buffer Overflow e Format String	0,1	10	1	baixo
Bypass controls	0,1	50	5	baixo
Memory leak	0,1	50	5	baixo
Race condition	0,1	10	1	baixo
TCP prediction, Spoofing e Hijacking	0,1	10	1	baixo
Man-in-the-middle	0,1	50	5	baixo
Replay	0,1	10	1	baixo
Fake authorization chain	0,1	10	1	baixo
Ponto único de falhas e vulnerabilidades	0,1	10	1	baixo
Ataque de inserção de conteúdo / código	0,1	10	1	baixo
Comprometimento da chave privada	0,1	50	5	baixo

Tabela 6.5 – Níveis de risco (probabilidade *x* impacto)

Com base em [83] os valores obtidos como nível de risco devem ser interpretados da seguinte maneira:

- risco baixo (valores entre 1 e 10): ações corretivas devem ser tomadas ou simplesmente o risco é assumido como aceitável.
- risco médio (valores no intervalo de 11 a 50): as ações corretivas devem ser planejadas e implantadas a curto ou médio prazo.
- risco alto (valores a partir de 51 e até 100): o sistema poderá até continuar operando, mas as medidas corretivas devem ser planejadas e implantadas o quanto antes possível, preferencialmente de imediato.

A negação de serviço com nível de risco de grau médio no protótipo implica em melhorias que podem ser agregadas futuramente ao sistema com alguma técnica de tolerância à faltas.

VIII. Recomendações de controle

Nesta fase são relacionados o conjunto de medidas a serem tomadas para minimizar o nível de risco. Como a gerência de risco foi aplicada ainda no ciclo de desenvolvimento do protótipo, admite-se que os controles considerados na seção 6.4.1, item IV, sejam suficientes para minimizar ou neutralizar as principais vulnerabilidades, ataques e violações identificadas.

Segundo a metodologia aplicada, algumas recomendações adicionais devem ser dirigidas aos administradores do sistema, em nosso caso, também ao cliente. A principal preocupação é com relação à gerência da chave privada dos principais.

O protótipo foi implementado utilizando os recursos do Java para geração e armazenamento de chaves; a base de chaves e a chave privada são protegidas por senhas. É muito importante frisar que senhas são susceptíveis a ataques de dicionário, força bruta e outros ataques relativamente simples de se executar. Logo, é importante utilizar senhas fortes – compostas de caracteres alfas-numéricos (minúsculos e maiúsculos) e caracteres especiais, tornando-as o menos previsíveis possível.

Com intuito de minimizar o efeito do comprometimento da chave privada, o principal não deve utilizá-la (chave *master*) em sua identificação no dia a dia. Ao invés disto, a chave *master* deve delegar a uma ou mais chaves secundárias (chave *slave*) o

direito de agir em nome do principal. As chaves *slaves* seriam utilizadas *online* e a chave *mast*er ficaria resguardada em local seguro e utilizada apenas *offline*. Assim, se em algum momento uma chave *slave* for comprometida só as cadeias que derivam da mesma estariam comprometidas. Caso a chave master seja utilizada no dia-a-dia e fique comprometida, isto significaria que o principal teria que renegociar todas as concessões relativas à mesma. Uma alternativa para evitar tal situação seria usar esquemas de *Blinded Keys* [97].

IX. Documentação dos resultados

Nesta fase são descritos: as vulnerabilidades, ameaças, ataques e violações identificadas, os controles adotados e as medidas para minimização do risco.

Como as vulnerabilidades foram descritas na seção 6.3.1 e os controles foram comentados na seção 6.4.1, item IV, os mesmos serão considerados como documentação desta fase.

6.4.2 Atenuação do Risco

As entradas para esta fase da metodologia são provenientes da fase anterior, seção 6.4.1, item VIII. Como a gerência de risco foi aplicada na fase de desenvolvimento do protótipo, este último já foi desenvolvido visando minimizar os tipos de vulnerabilidades identificadas.

Alguns cuidados adicionais necessários para dar imunidade ao risco ao sistema, que não envolvem diretamente o protótipo já foram comentados na seção 6.4.1 itens IV e VIII. Seguindo as recomendações da metodologia de gerência de riscos, reavaliações periódicas do protótipo devem ser realizadas, pois novas vulnerabilidades poderão ser identificadas com o passar do tempo. Além disto, quando a aplicação for completamente construída, o ambiente como um todo poderá ser avaliado pelo processo de gerência de risco.

Esta fase da metodologia busca priorizar, avaliar e implementar os controles apropriados para redução do risco. Considera-se que no projeto esta fase foi executada diversas vezes no processo de avaliações e correções por que passou o protótipo.

6.4.3 Considerações sobre os resultados do uso da metodologia Gerência de Risco

O uso da linguagem Java e de seu suporte de programação no desenvolvimento do protótipo pode ser considerado como estratégico na prevenção contra *buffer overflow*, *format string* e outros tipos de acessos ilegais a memória, por exemplo.

As avaliações e eliminações contínuas preconizadas pela metodologia Gerência de Risco no processo de desenvolvimento do protótipo evidentemente melhoram os resultados do projeto. Mesmo assim, foram detectadas algumas ineficiências na sistemática. Após uma análise automática do código através de uma ferramenta que utiliza injeção de faltas no sistema [98], observou-se principalmente deficiências relativas ao não tratamento de entradas inválidas. As partes do código onde tal tratamento não estava sendo feito foram reescritas e o teste refeito, não sendo mais identificadas falhas.

No projeto do protótipo foram empregadas técnicas de prevenção e de eliminação de vulnerabilidades. No item 6.3.3 são citadas técnicas de tolerância a vulnerabilidades. Cabe salientar que para *tolerar vulnerabilidades*, a solução adotada é geralmente baseada no uso de softwares adicionais. Isto não é uma tarefa muito simples e como conseqüência a complexidade do sistema pode ser aumentada, uma vez que novos elementos serão incorporados para minimizar as possíveis brechas do sistema como um todo. Porém, a inserção de softwares adicionais pode trazer vulnerabilidades ao sistema como um todo. Ou seja, se novos softwares forem adicionados todo o processo de gerência de risco deve ser refeito considerando este novo componente de software.

Outro fator importante a se considerar na escolha dos softwares para integrar um sistema é a opção (sempre) por aqueles amplamente utilizados no mercado, pois suas debilidades e características são também amplamente conhecidas.

Em geral pode-se considerar que as três técnicas de tratamento de vulnerabilidades (seção 6.3.3) foram utilizadas em diferentes níveis nas várias fases de desenvolvimento do protótipo. O resultado é uma arquitetura com um conjunto de características de segurança no mínimo interessantes.

6.5 Avaliação de performance

Os tempos de resposta para a aplicação de teste (seção 5.5), mostrados na Figura 6.1 foram medidos para avaliar o desempenho do protótipo quando o cliente busca cadeias de certificados de autorização em 3 casos:

- busca de certificado no repositório local;
- busca de certificados no repositório da federação *Home*, com ambos, o cliente e a federação pertencendo à mesma LAN (*Local Area Network*);
- busca de certificados nas federações associadas distribuídas através da Internet;

Nestes ensaios, a aplicação servidora limita-se a copiar a mensagem do fluxo de entrada para o fluxo de saída do *host* onde o serviço é executado; similar à função *echo* do protocolo ICMP. Porém, todas as funções de autenticação e autorização propostas neste trabalho são executadas, a exceção das negociações na construção de novas cadeias de autorização – sempre dependentes da aplicação considerada.

Analisando o tempo de resposta, em sua média, obtido com as medidas para acesso local (mesma máquina) e através na LAN, como mostra a Figura 6.1 e para acessos através da Internet (Figura 6.2) e, considerando os tamanhos de mensagem variando entre 256B e 1MB, então se comparado o tamanho da mensagem com o tempo de resposta, as seguintes tendências foram observadas:

a) para acessos local/LAN, comparando o (ORBacus + ORBAsec) com o (ORBacus + ORBAsec + modelo proposto) observa-se que, proporcionalmente:
   - a partir de 256B até 2KB, enquanto o tamanho da mensagem é incrementado em 8 vezes, o tempo de resposta é incrementado de 0,5 (a diferença no tempo de resposta entre ambos, passa de 6 vezes para 3 vezes – Figura 6.1). Isto significa que os incrementos no tamanho da mensagem não são diretamente proporcionais aos incrementos no tempo de resposta.
   - a partir de 2KB até 16KB ocorre o mesmo comportamento, enquanto a mensagem aumenta em 8 vezes o tempo de resposta é incrementado de 0,5 (a diferença no tempo de resposta, entre ambos passa de 3 para 1,5 vezes, Figura 6.1)
   - a partir de 16KB até 1MB a diferença no tempo de resposta entre ambos pode ser considerada insignificante.

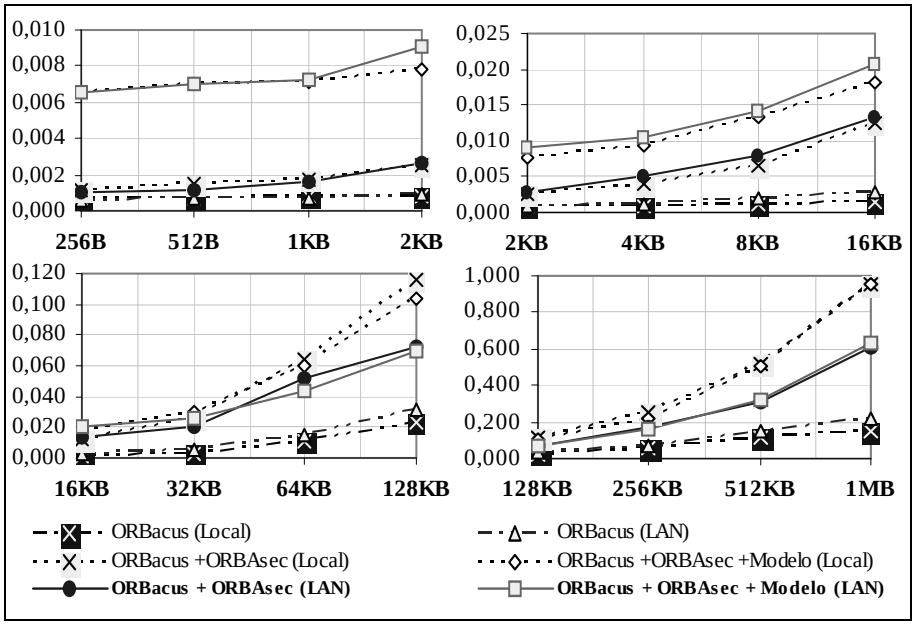


Figura 6.1 – Tempo médio de resposta (em segundos): acessos locais e em LAN

b) para acessos a Internet (Figura 6.2) comparando o (ORBacus + ORBAsec) com o (ORBacus + ORBAsec + modelo proposto) observa-se que, proporcionalmente:

- a partir de 256B até 2KB, quando o tamanho da mensagem é incrementado em 8 vezes, o tempo de resposta é incrementado de 0,2 (a diferença no tempo de resposta entre ambos passa de 2 vezes para 0,4 vezes).
- a partir de 2KB até 1MB a diferença no tempo de resposta entre ambos pode ser considerada insignificante.

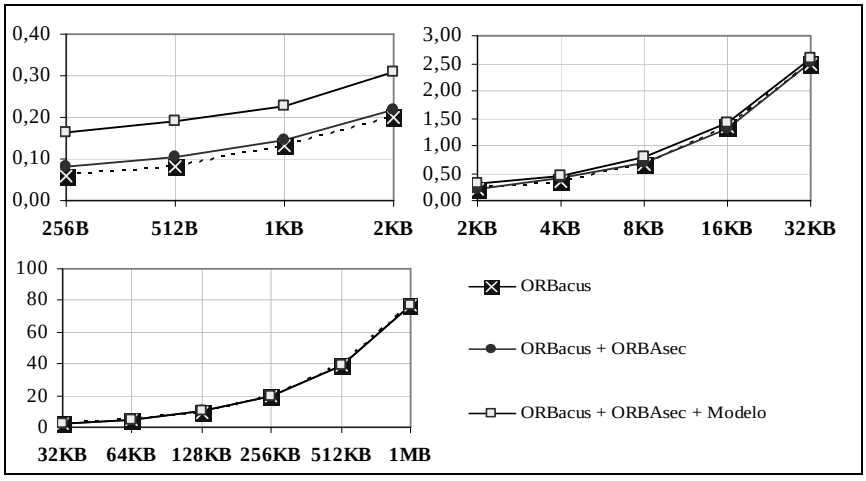


Figura 6.2 – Tempo de respostas médio (em segundos) para acessos pela Internet

Os resultados mostrados nas Figuras 6.1 e 6.2 foram obtidos utilizando PCs com processador de 1.4 MHz e 256 MB de memória RAM, rodando sistema operacional Linux (Kernel versão 2.4) e conectados em uma rede local *Fast Ethernet*. A conexão com a Internet, em seu ponto de menor largura de banda, está baseada em conexões ADSL com *upload* de 128Kbps. Os testes foram executados 1000 vezes e considerados em suas médias.

Observa-se que estes resultados experimentais podem variar significativamente com a aplicação construída usando o esquema proposto.

6.6 Considerações sobre o capítulo

O projeto e implementação conduzidos segundo uma sistemática de testes – a Gerência de Riscos – que considera as vulnerabilidades como inerentes ao sistema, permitiram o desenvolvimento de uma arquitetura com um grau de segurança melhorado.

A atribuição subjetiva (baseada na experiência técnica do avaliador) de probabilidades (Tabela 6.3) e de magnitudes de impacto (Tabela 6.4) é importante no processo de avaliação, porque o avaliador (perito em segurança) geralmente é quem define a política de segurança da corporação ou do serviço disponível na Internet. Logo, o mesmo tem a noção exata do grau de significância que cada elemento do seu sistema tem e, portanto, pode maximizar a relação custo/beneficio na escolha dos componentes de segurança.

Na prática qualquer método de avaliação, por mais formal que seja, se vale de premissas – que não incluem todas as situações atípicas / anômalas (ambiente com vulnerabilidades) possíveis. Logo, neste aspecto a gerência de riscos é mais efetiva porque preconiza a avaliação continuada de um sistema envolvendo desenvolvedores e administradores; só assim os espectros relativos a vulnerabilidades se estendem a níveis próximos da realidade do ambiente do sistema avaliado. Por ser uma sistemática continua, em função das análises, os avaliadores optam por sugerir novas alterações de configurações devido aos novos níveis de risco identificados.

Em geral a gerência de risco demanda, nas instituições interessadas, especialista com perfil profissional bastante compatível com o tipo de intrusão que a mesma pode sofrer, isto tende a melhorar a segurança em geral.

Na avaliação da performance do protótipo em linhas gerais, pode-se considerar que mensagens menores que 32KB sofrem atrasos no modelo proposto, principalmente devido ao protocolo *challenge*/*response* e as operações que envolvem a assinatura digital. A partir de tamanhos de mensagem de 32KB o tempo de resposta para o modelo proposto (ORBacus + ORBAsec + modelo proposto) mostra-se insignificantemente diferente de (ORBacus + ORBAsec).

Pelas considerações feitas na seção 6.4.1, item IV, percebe-se que o suporte de autenticação e autorização proposto no protótipo tem características bastante interessantes que reduzem a ocorrência, por concepção, de fraquezas comuns a outros sistemas de segurança.

6.7 Considerações finais

Neste capítulo foram abordadas as metodologias de testes para avaliação de segurança – CC e gerência de risco. Foi apresentado um exemplo de avaliação do protótipo através da gerência de risco. Foram considerados também aspectos relacionados ao desempenho do protótipo implementado.

7. Conclusão

Neste capítulo, por fim, será feito um balanço geral do trabalho. Neste sentido serão revisados os objetivos iniciais derivados da motivação do trabalho. Na seqüência, serão destacados os trabalhos realizados no período e serão sugeridos projetos futuros.

7.1 Revisão dos objetivos

O objetivo principal deste trabalho foi sugerir uma alternativa as soluções convencionais de serviços de autorização e autenticação normalmente encontradas em sistemas distribuídos e que fazem uso da rede mundial. Desejava-se encontrar soluções sem as dificuldades das soluções cadeias para estes serviços. As abordagens mais adaptadas a ambientes distribuídos em larga escala são fundamentadas em PKI. Porém, ainda que atendam as necessidades de um ambiente como a Internet, estas soluções dependem de serviços e de relações de confiança com entidades centralizadoras atendendo a um domínio de nomes.

Assumiu-se então, o uso do conceito de gerência de confiança adotando sua abordagem mais expressiva, o SDSI/SPKI. As relações de confiança, neste caso, passam a ser igualitárias e os principais identificados a partir de chaves públicas.

O objetivo geral neste trabalho foi o estudo envolvendo cadeias de confiança na autenticação e autorização em sistemas distribuídos de larga escala. Nesse sentido, foi desenvolvido um suporte integrando a especificação e a interpretação de políticas de autorização e, a geração de credenciais e seus relacionamentos. A ferramenta de suporte

trata a escalabilidade com base em propriedades do SDSI/SPKI – infra-estrutura orientada a chaves. Para alcançar o objetivo geral foram fixados alguns objetivos específicos:

1. Desenvolver um modelo de cadeias de confiança para autenticação e autorização que fornecesse meios para a localização automática de cadeias de certificados de autorização ligando um cliente ao servidor de aplicação.
2. Desenvolver mecanismos de suporte que permitam a negociação dinâmica da cadeia de certificados – levando ao serviço desejado – entre o principal e o detentor da mesma.
3. Desenvolver um suporte para autenticação e autorização para aplicações distribuídas na Internet. Este suporte deveria possuir meios de navegação e de associação de principais no sentido de facilitar a recuperação e a construção de cadeias de confiança apropriadas a sistemas de larga escala.
4. Adequar o suporte integrando-o a aplicações distribuídas na Internet. Conceber políticas de autorização próprias para as aplicações escolhidas.
5. Estudar técnicas de tolerância a faltas tomando como base *threshold subjects* do SDSI/SPKI.
6. Aplicar metodologias de testes para avaliar os resultados obtidos com o suporte de autenticação e autorização.

A maior parte destes objetivos foi alcançada no decorrer do trabalho. Evidentemente, não foi possível implementar tudo o que foi produzido conceitualmente e nem mesmo todos os algoritmos escritos durante o trabalho. Mas as idéias iniciais acabaram se expandindo e o que era esperado nos trabalhos de implementação tomou dimensões relativamente amplas. Por exemplo, no documento de qualificação, foi previsto o conceito de federação. A idéia da teia de federações foi concebida a posterior. A implementação de uma teia de federações envolveria muitos recursos e logo foi descartada. As implementações foram feitas considerando uma federação apenas.

Outro objetivo que inicialmente estava previsto era o desenvolvimento de uma aplicação de teste que evidenciasse as características do modelo de confiança proposto. A abrangência do trabalho tornou também inviável a realização desta atividade, em função da dependência seqüencial das atividades desenvolvidas; o algoritmo necessário para a navegação na teia de federações só foi concluído em 2003. O projeto que está em

andamento no DAS/UFSC pretende construir uma teia de federações, não em escala global, mas experimental e, em ambiente heterogêneo e distribuído.

Porém acredita-se que as contribuições conceituais suprem as dificuldades enfrentadas no cumprimento integral de alguns objetivos iniciais com relação a desenvolvimentos no protótipo.

7.2 Tarefas realizadas

Na viabilização deste trabalho uma série de etapas foi vencida até se alcançar os objetivos do mesmo, a seguir são enumerados as principais atividades desenvolvidas durante este período:

1. elaboração da proposta de trabalho;
2. definição do modelo de autenticação e autorização;
3. concepção do modelo que estende o SDSI/SPKI e define as noções de federação e de teia de federações. No modelo foram introduzidos também meios para a construção dinâmica de cadeias de autorização;
4. estudos das especificações padronizadas do SDSI/SPKI e de suas implementações disponíveis. Estudo das especificações CORBASec SL1 e SL2 e planejamento da implementação deste último (em nível de aplicação);
5. implementação do protótipo tomando como base o CORBASec SL2. Nesta implementação foram desenvolvidas: funções de uma biblioteca de codificação dos certificados SDSI/SPKI em XML e vice-versa, a tradução de certificados SPKI em X.509 e, uma implementação Java das bibliotecas do SDSI/SPKI e do XML e da base de armazenamento de certificados (Xindice);
6. elaboração da heurística de navegação pela teia de federações. Foi concebido o algoritmo de procura distribuído e implementado no contexto de uma federação; e
7. avaliação e análise do protótipo segundo a metodologia *Risk Management*. O estudo deste padrão e aplicação do mesmo na condução do projeto do protótipo ajudou as implementações a apresentar menos vulnerabilidades que provavelmente se teria sem o uso da mesma.

Uma listagem mais fina das tarefas executadas nas atividades citadas é apresentada abaixo:

- extensão do modelo administrativo do SDSI/SPKI, através da federação, facilitando a gerência dos certificado;
- definição de um esquema para criação de novas cadeias de autorização – que não é encontrada em nenhum dos trabalhos relacionados;
- definição de um esquema para formação da teia de federações dando escalabilidade as extensões apresentadas no modelo administrativo citado acima;
- concepção da heurística de navegação pela teia de federações, de formação arbitrária, sem auxílio do serviço de nomes, permitindo a localização de certificados de autorização no ambiente distribuído;
- sugestão de esquema de negociação de direitos de acesso, após a localização do certificado desejado, possibilitando a efetivação da delegação de direito de acesso;
- manutenção do suporte ao anonimato, em conformidade com as premissas do SPKI, alcançada através da isenção de obrigatoriedade de criação de cadastros/contas nos servidores de aplicação para poder acessá-los;
- manutenção do suporte a auditoria, mesmo com o suporte ao anonimato; a identificação do principal pode ser alcançada a partir da chave publica auditada em consultas a teia de federações;
- implementação da federação fazendo uso da integração do SDSI/SPKI ao CORBA permitindo a portabilidade de políticas de autorização através dos certificados armazenados em objetos do ambiente CORBA; e
- avaliação do protótipo, através de testes de desempenho – não mostrando acréscimos importantes ao tempo de processamento de mensagens.

7.3 Principais contribuições do trabalho

O modelo de confiança igualitário introduzido neste trabalho, estendendo o SDSI/SPKI e propondo-se a resolver os problemas identificados na literatura pode ser considerado a principal contribuição dos estudos do doutoramento. Além disto, o suporte

de autenticação e autorização implementando o modelo se mostrou flexível e seguro e, atendeu as expectativas em relação ao ambiente de larga escala.

As noções de federação e teia de federações parecem mais adequadas ao cenário assumido na proposta que as soluções encontradas na literatura associada. Estas noções propiciaram a criação de mecanismos que auxiliam na localização de principais e direitos de acesso em ambientes de larga escala.

A federação integrada a outras federações cumpre o papel de permitir a busca de direitos de acesso em ambientes formados arbitrariamente.

O modelo é livre das principais limitações dos sistemas clássicos. Além disto, oferece uma alternativa, não encontrada nos trabalhos relacionados, que é a heurística de navegação pela teia de federações para localizar direitos de acesso, negociar a concessão dos mesmos e criar cadeias de autorização não existentes.

O protótipo que integra o SDSI/SPKI ao ambiente CORBA constitui um passo importante na direção de evidenciar as características do modelo. Faltou a aplicação teste e a caracterização da teia de federações. Após o desenvolvimento da teia de federações será possível testar outras formas de integração das federações e outras heurísticas de navegação pela teia de federações.

Deste modo espera-se ter conseguido responder as perguntas formuladas na motivação e alcançado os objetivos colocados no início do trabalho.

7.4 Resultados

Além do protótipo e do projeto associado ao mesmo, pode-se destacar alguns dos resultados conseguidos na forma de artigos que foram publicados divulgando os resultados do trabalho:

- Santin, A. O., Fraga, J. S., Mello, E. R., Siqueira, F. TDSC-0016-0104 - Federation Web: A Scheme to Compound Authorization Chains on Large-Scale Distributed Systems. Submissão a IEEE Transactions on Dependable and Secure Computing. Em processo de revisão.

- Santin, A. O., Fraga, J. S., Mello, E. R., Siqueira, F. Federation Web: A Scheme to Compound Authorization Chains. In: The 22nd Symposium on Reliable Distributed Systems, 22nd IEEE-SRDS´2003, Florence, IT.
- Santin, A. O., Fraga, J. S., Maziero, C. Extending the SDSI/SPKI model through federation webs. In: Seventh IFIP TC-6 TC-11 CMS`2003, LNCS 2828, 2003, Torino, IT.
- Santin, A. O., Fraga, J. S., Mello, E. R., Siqueira, F. *Teias de Federações como Extensões ao Modelo de Autenticação e Autorização SDSI/SPKI*. In: XXI SBRC`2003, Natal - RN.
- Santin, A. O., Fraga, J. S., Mello, E. R., Siqueira, F. *Um modelo de autorização e autenticação baseado em redes de confiança para sistemas distribuídos de larga escala*. In: 4º SSI´02. São José dos Campos: CTA/ITA/IEC, 2002.
- Mello, E. R., FRAGA, J. S., SANTIN, A. O., SIQUEIRA, F.. *Building Trust Chains Between CORBA Objects* In: 1st LADC .2003. São Paulo - BR. LNCS 2847.
- Mello, E. R., Fraga, J. S., Santin, A. O. e Siqueira, F. *Redes de Confiança em Sistemas de Objetos CORBA.* 5º SSI`2003 - ITA - São José dos Campos – SP

7.5 Trabalhos futuros

No intuito de explorar as reais potencialidades do esquema proposto julgamos razoável a construção de uma teia de federações de caráter experimental para poder testar a eficácia do esquema proposto.

Deseja-se também poder desenvolver uma aplicação de teste e executar as metodologias de teste para validar o esquema na teia de federações experimental. Além disto, no desenvolvimento da aplicação o aspecto da negociação na concessão de direitos de acesso pode ser estudado com maior aprofundamento.

Julga-se importante, estudar e testar outras alternativas de composição da teia de federações, permitindo a experimentação de outras heurísticas de navegação que podem ser comparadas a que aqui foi proposta no intuito de melhorá-la ou substituí-la.

Anexo I. Controles Criptográficos

Anexo I.1 Considerações Iniciais

Um **criptosistema** (*cryptosystem*) ou sistema criptográfico define um par de transformações: cifragem e decifragem. A **cifragem** (*encryption*) é aplicada sobre os dados no formato original (em claro) para transformá-los em dados cifrados e a **decifragem** (*decryption*) é aplicada sobre os dados cifrados para torná-los em dados em claro novamente. Além dos dados, um valor "aleatório" independente denominado **chave** é usado em ambas as transformações.

Dependendo de como as chaves são usadas, o criptosistema pode ser simétrico ou assimétrico. Um criptosistema **simétrico** é aquele em que a mesma chave ou cópia desta é usada na cifragem e na decifragem dos dados. Neste caso, a chave deve ser guardada secretamente, o que faz com que o criptosistema simétrico também seja conhecido como **sistema de chave secreta**. Por outro lado, o criptosistema **assimétrico** faz uso de um par de chaves complementares, denominadas **chave privada** e **chave pública**. Assim, se a chave privada for utilizada para a cifragem, só a chave pública do par poderá ser utilizada para a decifragem. Analogamente, se a chave pública for utilizada para a cifragem, só a chave privada do par poderá ser utilizada para a decifragem. Porém, no criptosistema assimétrico, apenas a chave privada deve ser guardada secretamente, enquanto que a chave pública pode ser amplamente divulgada, por isso este criptosistema é também denominado de **sistema de chave pública**.

O restante do anexo compreende: gerência de chaves, algoritmos criptográficos e assinatura digital.

Anexo I.1 Gerência de chaves

Como visto acima, o segredo para manter o sigilo da informação cifrada é a chave, logo sua geração, distribuição e manutenção são muito importantes. A gerência de chaves tem por objetivo evitar a revelação não autorizada da chave e preservar sua integridade evitando a substituição, eliminação ou alteração não autorizada. Assim, o ciclo básico de vida de uma chave é o seguinte:

- Geração: consiste no processo de criação da chave, no qual dois aspectos são bastante importantes: a imprevisibilidade e a aleatoriedade do algoritmo de geração da chave e a escolha de bons parâmetros para tal.
- Distribuição: será tratada na seção Anexo I.1.1 devido a extensão do conteúdo.
- Manutenção: normalmente esta atividade recai sobre o usuário que é responsável por manusear a chave de maneira adequada e manter sua validade.

Anexo I.1.1 Distribuição de chaves

Várias soluções foram propostas para distribuir as chaves de maneira segura, tanto em sistema simétrico quanto em assimétrico. Dentre estas, pode-se destacar o sistema de trocas que sugere dois protocolos para comunicação segura com um servidor de autenticação e obtenção de chaves [RI-1]; um usando o sistema de chave secreta (Tabela I.1) e outro usando o sistema de chave pública (Tabela I.2).

Cabeçalho	**Mensagem**
$\underline{A} \rightarrow \underline{S}$	$\underline{A}, \underline{B}, N_{\underline{A}}$
$\underline{S} \rightarrow \underline{A}$	$\{ N_{\underline{A}}, \underline{B}, K_{\underline{A}+\underline{B}}, \{ K_{\underline{A}+\underline{B}} , \underline{A} \}_{K^{\underline{B}}} \}_{K^{\underline{A}}}$
$\underline{A} \rightarrow \underline{B}$	$\{ K_{\underline{A}+\underline{B}} , \underline{A} \}_{K^{\underline{B}}}$
$\underline{B} \rightarrow \underline{A}$	$\{ N_{\underline{B}} \}_{K^{\underline{A}+\underline{B}}}$
$\underline{A} \rightarrow \underline{B}$	$\{ N_{\underline{B}} - 1 \}_{K^{\underline{A}+\underline{B}}}$

Tabela I.1- Distribuição da chave secreta baseada num servidor de autenticação

Legenda

$\mathbf{N}_{\underline{x}}$	N (*nonce*) é um número inteiro enviada na mensagem por $\underline{x}$ para demonstrar que esta é recente.
$\{ M \}_{K^{\underline{x}}}$	Significa que a mensagem *M* foi cifrada com a chave secreta (k) de $\underline{x}$.
$\{ M \}_{K^{\underline{x+y}}}$	Significa que a mensagem *M* foi cifrada com a chave de sessão (k) entre $\underline{x}$ e $\underline{y}$.
$\mathbf{K}_{x+y}$	Indica uma chave de sessão (K) entre $\underline{x}$ e $\underline{y}$. A **chave de sessão** é temporária e dura só uma sessão.

Cabeçalho	Mensagem
A → S	A, B
S → A	$\{ \mathbf{PK_B}, B \}_{SK}^{S}$
A → B	$\{ N_A, A \}_{PK}^{B}$
B → S	B, A
S → B	$\{ \mathbf{PK_A}, A \}_{SK}^{S}$
B → A	$\{ N_A, N_B \}_{PK}^{A}$
A → B	$\{ N_B \}_{PK}^{B}$

Tabela I.2- Distribuição da chave pública baseada num servidor de autenticação

Legenda

$\mathbf{N}_x$	N (*nonce*) é um número inteiro introduzido na mensagem enviada por x, para demonstrar que esta é recente.
$\{ M \}_{SK}^{x}$	Significa que a mensagem *M* foi cifrada com a chave privada (SK) de x.
$\{ M \}_{PK}^{x}$	Significa que a mensagem *M* foi cifrada com a chave pública de (PK) de x.
$\mathbf{PK}_x$	Indica a chave pública (PK) de x.

Os protocolos permitem que um principal[30] num recurso[31] A, requisite a um servidor de autenticação S uma chave para se comunicar com um principal num recurso B.

Anexo I.2 Algoritmos Criptográficos

Os algoritmos criptográficos que implementam os criptosistemas simétricos baseiam sua segurança nos princípios da **confusão** (*confusion*) e da **difusão** (*diffusion*). A confusão para eliminar relações óbvias entre os bits de entrada e os de saída, e a difusão para produzir um efeito de espalhamento dos dados no formato original (em claro) sobre os dados cifrados. Para alcançar tais resultados os algoritmos utilizam tabelas, baseados nas quais aplicam transformações e permutações nos dados por um determinado número de vezes. O *DES* (*Data Encryption Standard*), por exemplo, um dos algoritmos mais difundidos atualmente é implementado desta maneira.

O espaço de solução computacional de um problema matemático pode ter tempo previsível (*P*) ou imprevisível (*NP*). Assim, os algoritmos criptográficos para

[30] Um principal é o verdadeiro detentor de uma identidade, neste contexto pode ser um usuário, um sistema, um processo ou uma combinação destes que represente uma entidade ativa – executa uma ação sobre um recurso.

[31] Um recurso é um objeto ou um dispositivo (memória, arquivo, CPU, ...) que representa uma entidade passiva – sobre uma ação executada por um principal.

criptosistemas assimétricos baseiam sua segurança em **problemas matemáticos complexos** (que são *P* no cálculo de uma função matemática e *NP* no cálculo da inversa da mesma função). Um exemplo deste tipo de função é a fatoração de inteiros primos, onde é *P* escolher dois números primos grandes (a,b > 10^{12}) e multiplicá-los para obter um produto (c). Porém, é *NP* fatorar o produto c e chegar aos mesmo dois números inteiros primos (a? e b?) que geraram c. O exemplo mais comum de emprego desta técnica é o algoritmo *RSA* (*Rivest, Shamir* e *Adleman*) largamente utilizado nos dias atuais. Maiores detalhes sobre o assunto podem ser encontrados em [RI-2].

É importante salientar que os algoritmos que implementam criptosistemas de chave simétrica exigem um esforço computacional bem menor para serem executados que os implementados no criptosistema assimétrico. Porém, os simétricos têm a desvantagem de necessitar o compartilhamento da chave secreta para poderem ser utilizados. Logo, é muito comum o uso de um sistema híbrido na troca de mensagens, onde são utilizados os algoritmos do sistema de chave pública para a distribuição das chaves secretas (chaves de sessão) que serão utilizadas para cifrar os dados em trânsito, obtendo-se assim o melhor das características de cada um dos sistemas.

Anexo I.3 Assinatura Digital

A assinatura digital (*AD*) é um mecanismo que implementa a versão eletrônica da assinatura de punho. A forma mais comum de implementação de *AD* é a geração de um sumário que é cifrado com a chave privada do remetente, gerando um apêndice que é enviado anexado a mensagem assinada (Figura I.1).

Para gerar o sumário é utilizada uma função de *hash H*, dada por $h = H(m)$, onde *H* retorna sempre uma *string* de tamanho fixo – *h* (normalmente de 128 ou 160 bits), independente do tamanho da mensagem (*m*). Como o espaço de conteúdo da mensagem *m* é bastante amplo é imperativo a escolha de um algoritmo de geração de sumário que empregue uma função de *hash* com a menor taxa de colisão[32] possível.

[32] Geração de um mesmo sumário a partir de mensagens diferentes.

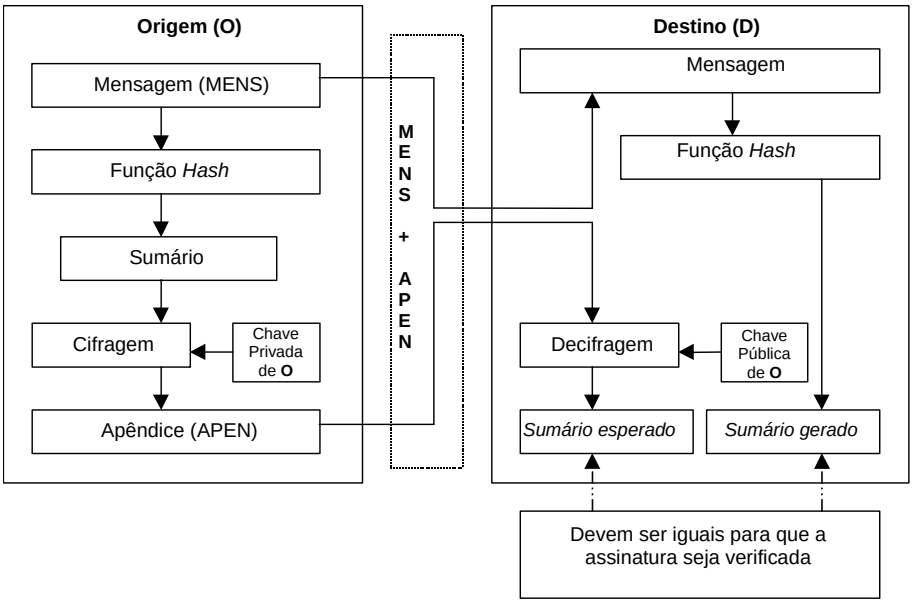


Figura I.1 - Processo de assinatura digital

Anexo I.4 Considerações finais

Neste anexo foram abordados brevemente alguns dos fundamentos relacionados a controles criptográficos: tipos de criptosistemas, gerência de chaves, tipos de algoritmos criptográficos e o mecanismo de assinatura digital.

Anexo I.5 Referências Bibliográficas

[RI-1] – Needham, R. M. e Schoeder, M. D. *Using Encyiption for Authentication in Large Networks of Computer*, *Comunication of de ACM*, Vol. 21, Nº 12, pag. 993-999, dezembro de 1978.

[RI-2] – Menezes, A. J., Oorschot, P. C. e Vanstone, S. A. *Handbook of Applied Cryptography*, ISBN: 0-8493-8523-7, *CRC Press*, 1999. http://cacr.math.uwaterloo.ca/hac/.

Anexo II. Kerberos Network Authentication Service

Anexo II.1 Considerações iniciais

O *Kerberos*[33] foi desenvolvido em meados dos anos 80 como parte do projeto *Athena* no *Massachusetts Institute of Technology* [RII-1]. O modelo adotado pelo *Kerberos* é o proposto no protocolo de autenticação com confiança num terceiro (*trusted third-party*), proposto por *Needham* e *Schroeder* (Anexo I, Tabela I.1) e em modificações propostas por *Denning* e *Sacco*, conforme a *RFC1510* [RII-2].

Neste anexo serão mostrados os componentes fundamentais da infra-estrutura *Kerberos* e a maneira como os mesmos se inter-relacionam.

Anexo II.2 Infra-estrutura Kerberos

O servidor *Kerberos* é conhecido como *Key Distribution Centre* (*KDC*) e oferece um *Authentication Service* (*AS*) e um *Ticket Granting Service* (*TGS*) a seus clientes. Para usufruir da infra-estrutura o cliente (*C*) inicialmente deve trocar uma mensagem com o *AS* para se autenticar. Uma vez autenticado o cliente tem uma credencial com um período de validade, denominada *Ticket Granting Ticket* (*TGT*) para se comunicar com o *TGS* e obter uma credencial para apresentar ao servidor de aplicação (*S*). Enquanto o *TGT* for válido, o cliente pode solicitar quantas credenciais forem necessárias, uma por sessão, para

[33] http://web.mit.edu/kerberos/www

comunicar-se com os servidores (*S*). Na Tabela II.1 será mostrado um resumo das mensagens trocadas (Figura II.1) para detalhar o protocolo usado pelo *Kerberos*.

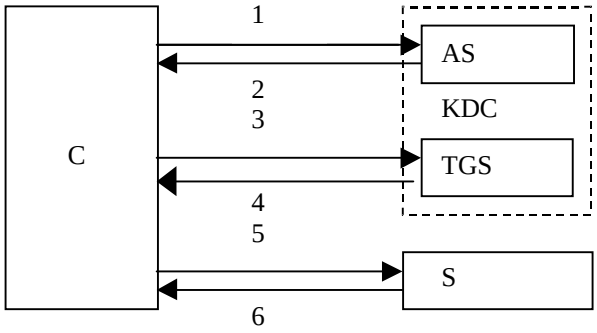


Figura II.1 - Troca de mensagens para autenticação no Servidor Kerberos

Troca	**Descrição**
1	Essa mensagem é conhecida como pedido inicial de autenticação
2	Se C passar pela autenticação local, possuirá um *TGT*
3	C pode solicitar ao TGS um *ticket* e uma chave de sessão para acessar *S*
4	Enquanto o *TGT* for válido, C pode fazer quantas solicitações (como a do passo 3) desejar, uma para cada sessão com o servidor *S*
5	*C* envia para *S* um *authenticator*, um *nonce* e uma Solicitação de Serviço (*SS*) em claro ou cifrada
6	Opcionalmente, *S* responde com autenticação do servidor

Troca	**Cabeçalho**	**Mensagem**
1	C → AS	C, TGS, n
2	AS → C	$\{\mathbf{K_{C+TGS}}, n\}_K^C$, $\{C, TGS, t1, t2, \mathbf{K_{C+TGS}}\}_K^{TGS}$
3	C → TGS	$\{C, t\}_K^{C+TGS}$, $\{C, TGS, t1, t2, \mathbf{K_{C+TGS}}\}_K^{TGS}$, S, n1
4	TGS → C	$\{\mathbf{K_{CS}}, n1\}_K^{C+TGS}$, $\{C, S, t1', t2', \mathbf{K_{C+S}}\}_K^{S}$
5	C → S	$\{C, t'\}_K^{C+S}$, $\{C, S, t1', t2', \mathbf{K_{C+S}}\}_K^{S}$, n2, SS
6	S → C	$\{n2\}_K^{C+S}$

Tabela II.1 – Conteúdo e significado das troca de mensagens do protocolo *Kerberos*

Legenda

$\{C, t^z\}_K^{C+S}$	é o *autenticador* de C para com S, que é usado uma única vez - porque as chave de sessão são usadas uma para cada sessão, assim como o *timestamp* (t^z).
$\{M\}_K$	significa que a mensagem *M* foi cifrada com a chave k.
$\mathbf{K_{xy}}$	indica uma chave de sessão **K** entre **x** e **y**.
$\mathbf{K}_x$	indica a chave secreta **K** de x.
$\mathbf{n}_x$	**n** é um número inteiro introduzido na mensagem enviada por x, para demonstrar que a mesma é recente, *nonce*.
W	indica um principal W.
$\mathbf{t}x^z$	indica tempo inicial ou final.
$\mathbf{t^z}$	indica um *timestamp*.
$\mathbf{K^x}$	indica a chave secreta do(s) principal(is)

O servidor *Kerberos* oferece além do serviço de autenticação, o serviço de comunicação com integridade (KRB_SAFE) e/ou com confidencialidade (KRB_PRIV).

A infra-estrutura mostrada acima deve estar disponível em cada *KDC* ou 'domínio *kerberos'* (*realm*). Num *realm*, um principal pode ser: um nome de usuário, um serviço, um *host* ou outro identificador único.

Para permitir a interoperabilidade o *Kerberos* adota formatos padrão (domínio e *X.500*) para a composição do nome que identifica um objeto num *realm*; além de suportar outros formatos que obedecem a certa regra de formação, como pode ser visto nos exemplos abaixo:

- domínio: *host.subdomínio.domínio*
- *X.500*: *C=US/O=OSF*
- *other*: *NAMETYPE:* restante do nome (sem restrições)
- *reserved*: reservada, mas não conflitante com os formatos acima.

É importante observar que para ganhar escalabilidade, através da execução de operações inter-domínio o administrador de um *realm* local deve registrar o seu *realm* num *TGS* remoto (*RTGS*) para compartilharem uma chave inter-domínio. Assim, é possível um principal acessar um serviço que se encontra num *realm* (remoto) diferente do que está registrado (local), solicitando ao *AS* local um *TGT* para apresentar ao *RTGS* e obter um *ticket* que utilizará no acesso ao serviço remoto.

Anexo II.3 Considerações finais

Neste anexo foram abordados brevemente os componentes da infra-estrutura do *Kerberos*, mostrando as principais trocas envolvidas no inter-relacionamento destes componentes, incluindo os clientes, bem como a nomenclatura adotada para permitir interações inter-domínio e o esquema que permite escalabilidade.

Anexo II.4 Referências bibliográficas

[RII-1] - *Massachusetts Institute of Technology*. *Project Athena Technical Plan*. ftp://athena-dist.mit.edu/pub/kerberos/doc/techplan.PS, 1985.

[RII-2] - *The Internet Engineering Task Force*, *Kerberos Network Authentication Service (Versão 5)*. http://www.ietf.org/rfc/rfc1510.txt, 1993.

Anexo III. Infra-estrutura de chave pública

Anexo III.1 Considerações iniciais

A infra-estrutura de chave pública (*Public Key Infrastructure - PKI*) é o resultado de um estudo encomendado pelo *NIST* (*National Institute of Standards and Technology*) ao *MITRE Corporation*, publicado em abril de 1994 [RIII-1]. Tal estudo teve por objetivo identificar alternativas para automatizar a gerência de chaves públicas e respectivos certificados para o governo americano (EUA), com intuito de tornar viável também o comércio eletrônico seguro.

Objetivando padronizar as várias versões de *PKI*, disponibilizadas por diferentes empresas, em 1997 o *NIST* em parceria com um grupo destas empresas, publicaram o [RIII-2] para promover a interoperabilidade entre as *PKI*. O *NIST* mantém até atualmente um programa, *NIST PKI Program*, para estudar e atualizar a *PKI*. Maiores informações podem ser obtidas em *http://csrc.ncsl.nist.gov/pki/*. A seguir serão mostrados os principais componentes da infra-estrutura *PKI*.

Anexo III.2 Infra-estrutura PKI

A infra-estrutura de chave pública suporta duas operações básicas, a **certificação** (*certification*) e a **validação** (*validation*), onde a certificação é o processo de associação de uma chave pública a uma entidade e a validação é o ato de verificação da validade da certificação. Atualmente na certificação pode-se associar um principal a um certificado que

contém um conjunto de informações sobre o mesmo. Maiores detalhes sobre os dados que um certificado contém podem ser obtidas no Anexo IV.3. Porém, quando um certificado não for mais válido este é relacionado numa lista de certificados revogados (*CRL*) [RIII-3]; uma espécie de lista negra. Tais dados, certificados e *CRL*s devem ser armazenados num repositório. Neste caso, o repositório representa um banco de dados capaz de recuperar os certificados e *CRL*s sem exigir a autenticação do solicitante. Este repositório pode ser implementado, por exemplo, num serviço de diretório como o da recomendação *X.500* da *ITU-T* (*International Telecommunications Union - Telecommunication Standardization Sector*).

Para facilitar a exposição do assunto será adotada a abordagem [RIII-2] que especifica a infra-estrutura mínima de componentes para prover interoperabilidade em *PKI*s: autoridade certificadora, autoridade de registro organizacional e clientes *PKI*.

Anexo III.2.1 Autoridade Certificadora (Certification Authority – CA)

A *CA* é responsável pela geração, arquivamento, publicação e revogação de certificados. Após a geração o certificado é arquivado e publicado através do serviço de diretório, que armazenará também as *CRL*s que a *CA* emitir. Além disto, para ganhar escalabilidade a *CA* pode estabelecer relações de confiança que compõem uma estrutura hierárquica ou uma rede (cadeia) de confiança. Se as relações de confiança formam uma estrutura hierárquica, a confiança é delegada por uma *CA* quando esta certifica uma *CA* subordinada, sendo que a delegação começa na *CA* raiz – entidade de confiança de qualquer membro da estrutura. Quando, porém, as relações de confiança se derem aos pares (*peer*) formarão uma rede de confiança por certificação cruzada (*cross-certification*) que permitirá a composição de múltiplos caminhos de confiança (*trust paths*) entre duas *CA*s quaisquer.

Anexo III.2.2 Autoridade de Registro Organizacional (Organizational Registrtion Authority – ORA)

A *ORA* é a entidade da *PKI* a qual o usuário (cliente) efetivamente tem acesso. A ORA atua intermediando o pedido de certificação do usuário, validando os dados fornecidos na solicitação e quando for necessário até requisitando informações

complementarem ou comprovações que podem chegar a ter caráter presencial. Após a validação de tais informações, a *ORA* assina o pedido de certificação e o envia a *CA* em nome do usuário solicitante. A *CA*, após aceitar o pedido, gera um certificado que envia a *ORA* e ao serviço de diretório. Além de atender as solicitações para certificação, a *ORA* também é responsável pela geração do pedido de revogação de certificados feitos, quando necessário, à *CA*.

Anexo III.2.3 Clientes PKI

Basicamente a *PKI* tem dois tipos de clientes, o detentor de certificado *(Certificate Holder*) e o verificador (*verifier*). Os detentores de certificados podem ser: *ORA*, *CA* ou outra entidade deste tipo, para as quais *PKI* oferece funções de gerência de certificados como as comentadas na seção Anexo III.2.2. Já aos verificadores, que podem ser: *ORA*, usuários, sistema etc, a *PKI* prevê acesso de consultar (*query*) ao repositório para recuperar certificados de entidades e/ou a última *CRL* emitida por uma *CA*.

Anexo III.3 Considerações finais

Neste anexo foram expostos brevemente os integrantes fundamentais que compõem uma infra-estrutura de chave pública e suas respectivas principais funções.

Anexo III.4 Referências bibliográficas

[RIII-1] - *MITRE Corporation, Public Key Infrastructure Study – Final Report*. http://csrc.nist.gov/pki/documents/mitre.ps, 1994.

[RIII-2] - *National Institute of Standards and Technology, Minimum Interoperability Specifications for PKI Components, Version 1*. http://csrc.ncsl.nist.gov/pki/documents/mispcv1.ps, 1997.

[RIII-3] - *The Internet Engineering Task Force. Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. http://www.ietf.org/rfc/rfc2459.txt, 1999.

Anexo IV. Framework de Autenticação X.509

Anexo IV.1 Considerações iniciais

A *ITU-T*[34] tem um conjunto de recomendações denominadas série *X.500* que compreendem serviços de diretório (*directory*) de maneira geral [RIV-1]; onde o protocolo de acesso ao diretório utilizado para recuperar e armazenar informações é o *DAP* (*Directory Access Protocol - X.519*). Tais informações estão dispostas em uma estrutura formando uma árvore hierárquica denominada *DIT* (*Directory Information Tree - X.501*), cada objeto (nó folha) da árvore é identificado unicamente pelo caminho percorrido para chegar até o mesmo (*DN – Distinguished Name*), a partir do nó raiz (Figura III.1). Assim, através da *DIT* é possível identificar unica e globalmente um objeto, se considerada a árvore compondo a distribuição geográfica do planeta. Se for considerada agora a infra-estrutura de chave pública, *PKI* (Anexo III), implementada em um diretório *X.500*, conseqüentemente se terá a *CA*, *ORA* e verificadores como objetos da *DIT* que podem ser identificados por um *DN*. Sabendo-se que a recomendação *X.509* especifica a autenticação forte (*strong authentication*) usando *PKI* sobre um diretório *X.500*, pode-se então definir *X.509* como sendo um serviço de autenticação implementado em um serviço de diretório X.500 através de PKI.

Neste anexo será discorrido sobre os mecanismos que permitem autenticação na infra-estrutura *X.509*, assim como serão mostrados os campos de um certificado *X.509 v3*.

---

[34] *International Telecommunication Union - Telecommunication Standardization Sector.*

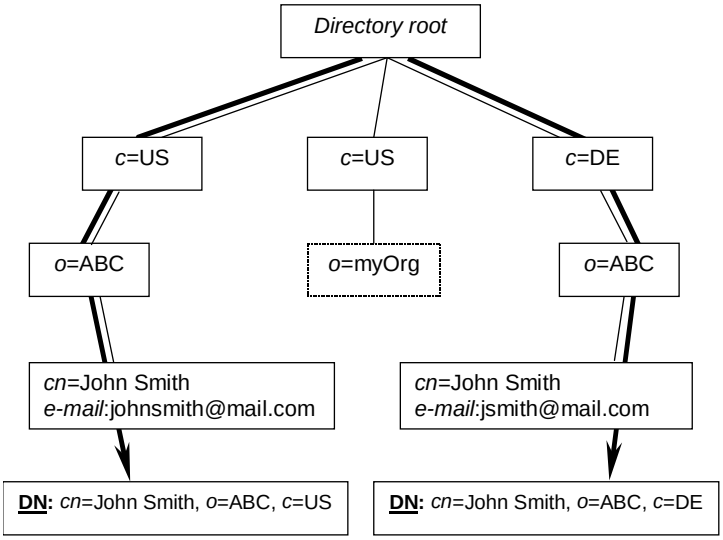


Figura IV.1 – Estrutura do diretório *X.500*

Anexo IV.2 Infra-estrutura X.509

O *framework X.509* consiste em autenticação simples (*simple authentication*) – através de um *DN* e uma *password* e, autenticação forte (*strong authentication*) – usando *PKCS* (*Public Key Cryptosystem*). Sabendo que a autenticação simples é destinada a atender necessidades restritas, será considerada neste contexto apenas a autenticação forte. Assim, o ambiente considerado é constituído de *PKCS* em *PKI* sobre *X.500*, o que resulta nas seguintes premissas: a *CA* e os principais são unicamente identificados na *DIT*, cada principal tem uma chave privada e um certificado emitido por uma *CA* em seu poder. Então, um principal pode assinar digitalmente mensagens que serão validadas pelo verificador (destinatário da mensagem).

A seguir serão mostradas as mensagens trocadas nos casos de autenticação: *one-way* (Tabela IV.1), *two-way* (Tabela IV.2) *e three-way* (Tabela IV.3).

Cabeçalho	**Mensagem**	**Comentários**
1. $\underline{A} \rightarrow \underline{B}$	$\{\underline{B}, t_a, n_a, x_a, y_a\}_{S\underline{A}}$	O campo x_a, se usado, expressa a autenticação da origem. O y_a, quando usado, contém chaves criptografadas com a chave pública de $\underline{B}$, para uso posterior.
2. O principal $\underline{B}$ obtém a chave pública do principal $\underline{A}$ do diretório e verifica se o certificado não expirou, verifica a assinatura digital, verifica se é o destinatário, verifica se o *timestamp* é atual e se o n_a não é retransmissão.		

Tabela IV.1– One-way authentication exchanges

Cabeçalho	Mensagem	Comentários
1. $\underline{A} \rightarrow \underline{B}$	$\{\underline{B}, t_a, n_a, x_a, y_a\}_{S\underline{A}}$	O campo x_a, se usado, expressa a autenticação da origem. O y_a, quando usado, contém chaves criptografadas com a chave pública de B, para uso posterior.
2. O principal B obtém a chave pública do principal A do diretório e verifica se o certificado não expirou, verifica a assinatura digital, verifica se é o destinatário, verifica se o *timestamp* é atual e se o n_a não é retransmissão.		
3. $\underline{B} \rightarrow \underline{A}$	$\{\underline{A}, t_b, n_a, n_b, x_b\}_{S\underline{B}}$	O campo x_b, se usado, expressa a autenticação da origem. O y_b, quando usado, contém chaves criptografadas com a chave pública de A, para uso posterior.
4. O principal A verifica a assinatura digital, verifica se é o destinatário, verifica se o *timestamp* é atual e se o n_b não é retransmissão.		

Tabela IV.2 – Two-way authentication exchanges

Cabeçalho	Mensagem	Comentários
1. $\underline{A} \rightarrow \underline{B}$	$\{\underline{B}, t_a, n_a, x_a, y_a\}_{S\underline{A}}$	O campo x_a, se usado, expressa a autenticação da origem. O y_a, quando usado, contém chaves criptografadas com a chave pública de B, para uso posterior.
2. O principal B obtém a chave pública do principal A do diretório e verifica se o certificado não expirou, verifica a assinatura digital, verifica se é o destinatário e se o n_a não é retransmissão. O *timestamp* geralmente é zero neste tipo de autenticação.		
3. $\underline{B} \rightarrow \underline{A}$	$\{\underline{A}, t_b, n_a, n_b, x_b, y_b\}_{S\underline{B}}$	O campo x_b, se usado, expressa a autenticação da origem. O y_b, quando usado, contém chaves criptografadas com a chave pública de A, para uso posterior.
4. O principal A verifica a assinatura digital, verifica se é o destinatário, se o n_b não é retransmissão e se o n_a recebido é igual ao enviado anteriormente. O *timestamp* geralmente é zero neste tipo de autenticação.		
5. $\underline{A} \rightarrow \underline{B}$	$\{\underline{B}, n_b\}_{S\underline{A}}$	Este passo adicional é utilizado em substituição a utilização de *timestamp*.
6. O principal B verifica a assinatura digital, e se o n_b recebido é igual ao enviado anteriormente.		

Tabela IV.3 – Three-way authentication exchanges

Legenda

$\mathbf{n}_{\underline{x}}$ **n** é um número não repetido introduzido na mensagem enviada por x, para demonstrar que a mesma é recente, *nonce*.

$\{M\}_{S\underline{K}}$ indica que a mensagem *M* foi assinada digitalmente por k.

$\mathbf{t}_x$ indica um *timestamp* gerado por x, que consiste de uma ou duas datas.

Anexo IV.3 Certificados X.509 versão 3

Apenas a título de ilustração, na Figura IV.2 são mostrados os campos que compõem um certificado *X.509 versão 3*. Cabendo salientar que geralmente os campos denominados *Basic Certificate Fields* são utilizados para propósito de autenticação e os campos *Certificate Extension Fields* para fins de autorização. Maiores detalhes podem ser obtidos em [RIV-2].

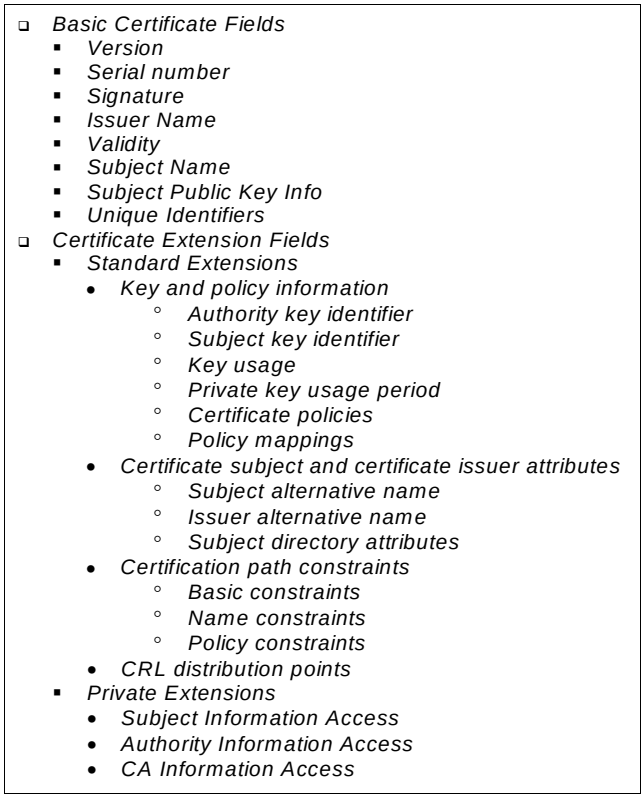


Figura IV.2 - Campos de um certificado *X.509 v3*

Anexo IV.4 Considerações finais

Neste anexo foram mostradas brevemente as principais características e os mecanismos envolvidos na autenticação baseada na recomendação *X.509*, assim como os campos que compõem um certificado *X.509*. Salientando que a versão de *X.509* utilizada neste contexto data de agosto de 1997, quando denominava-se *Framework de Autenticação X.509*. Porém, a partir de março de 2000 essa recomendação passou a se chamar *Public-key and attribute certificate frameworks* [RIV-3]. Observa-se ainda que além da *ITU-T*, o *Internet Engineering Task Force* mantém um grupo trabalhando na especificação de *X.509* para Internet [RIV-4].

Anexo IV.5 Referências bibliográficas

[RIV-1] - *International Telecommunication Union, Telecommunication Standardization Sector - Publications. Recommendation X.509 – Information technology – Open*

Systems Interconnection – The Directory Authentication Framework, november 1993.

[RIV-2] - *The Internet Engineering Task Force. Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. http://www.ietf.org/rfc/rfc2459.txt, 1999.

[RIV-3] - *International Telecommunication Union, Telecommunication Standardization Sector - Publications*. http://www.itu.int/publications/telecom.htm, 2001.

[RIV-4] - *The Internet Engineering Task Force, Public-Key Infrastructure (X.509) (pkix)*. http://www.ietf.org/html.charters/pkix-charter.html, 2001.

Anexo V. Arquitetura CORBA (*Common Object Request Broker Architecture*)

Anexo V.1 Considerações iniciais

A arquitetura *CORBA* [RV-1], atualmente na versão 2.6, define um ambiente para objetos distribuídos, dividindo o espaço de objetos em três: objetos de aplicação – construídos pelo programador da aplicação, objetos de serviço – (*Commom Object Services Specification – COSS*) oferecendo suporte aos serviços comuns independentes de domínios de aplicação e as facilidades comuns (*Common Facilities*) constituindo serviços orientados aos domínios de aplicação.

Os principais componentes da arquitetura CORBA [RV-2] são mostrados na Figura V.1, onde:

- *Object*: é uma entidade programável CORBA, consistindo de uma identificação, uma interface e uma implementação.
- *Servant*: é visível apenas no servidor, representa um recurso de memória ou a *CPU* trabalhando para executar uma operação em um objeto.
- *Client*: é a entidade que invoca uma operação em um objeto de implementação. Idealmente a invocação deve ser tão simples quanto chamar um método num objeto local.
- *Object Request Brocker* (*ORB)*: provê um mecanismo de comunicação transparente entre o cliente e a implementação do objeto. Esconde a complexidade do ambiente distribuído. Quando o cliente invoca uma operação é responsável por encontrar a implementação do objeto, fazer a sua ativação, enviar-lhe a requisição e retornar o

resultado da operação ao cliente, de maneira transparente. A comunicação entre *ORB*s é feita através do protocolo *General Inter-ORB Protocol* (*GIOP*), que tem no *Internet Inter-ORB Protocol (IIOP)* – versão do *GIOP* para a pilha *TCP* / *IP* – a chave para interoperabilidade entre objetos escritos em diferentes linguagens.

- *ORB Interface:* define uma interface abstrata para o *ORB*, utilizada para conversões e invocação dinâmica – *DII* / *DSI* utilizadas por *browser/databases*, por exemplo, porque nos mesmos seria impraticável o *linking* estático de todas as classes possíveis de *stubs / skeletons*.

- *IDL stubs and skeletons:* serve para ligar o cliente (*stub*) e o servidor (*skeleton*) de aplicação entre si através do *ORB*.

- *Dynamic Invocation Interface(DII)*: é uma interface utilizada para acesso direto (sem *stubs*) aos mecanismos oferecidos pelo *ORB*.

- *Dynamic Skeleton Interface (DSI)*: permite ao *ORB* enviar requisições para a implementação de um objeto sem conhecer o tipo de objeto implementado (*sem skeleton*).

- *Object Adapter:* associa a implementação do objeto ao *ORB*, auxiliando-o no envio e ativação do objeto.

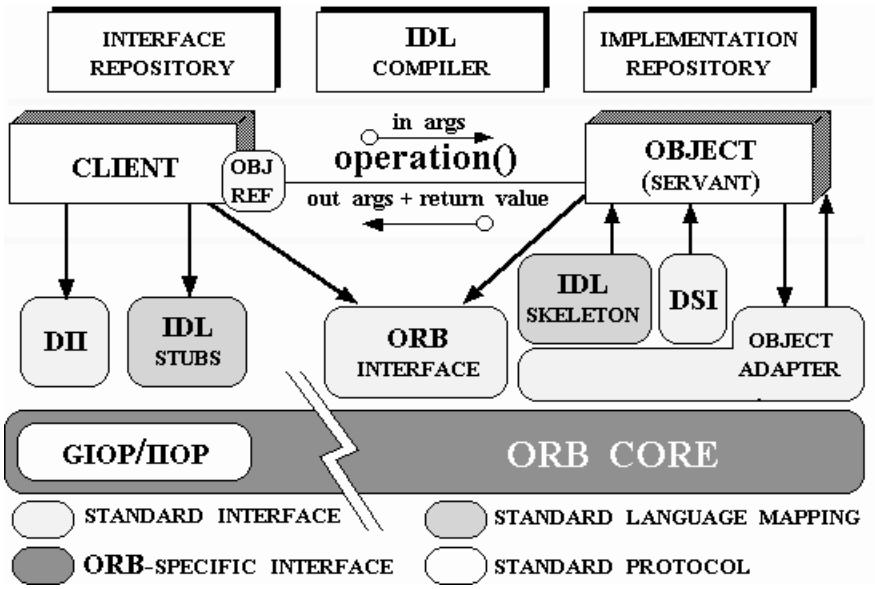


Figura V.1 – Arquitetura CORBA [RV-1]

O CORBA assume que toda a dinâmica do sistema é dada por requisições feitas através da *OMG Interface Definition Language* (OMG *IDL*) [RV-3], passando pelo *ORB* para chegar a implementação do objeto, Figura V.2.

Neste anexo será enfatizado o modelo CORBA de segurança (CORBASec), logo, apenas descreveremos simplificadamente com cunho ilustrativo, como seria a dinâmica de funcionamento do ambiente na invocação de uma operação em um objeto. Para executar a operação o cliente deveria obter uma referência para o objeto (*object reference*) no serviço de nomes - Cos*Naming*, por exemplo. Em seguida precisaria fazer o *binding* com a referência ao objeto. Então, no *stub* será feito o *marshalling* (serialização) dos argumentos, enviados pela rede através do *GIOP/IIOP* para o *skeleton* no lado do servidor, onde é executado o *unmarshalling* (deserialização) e os argumentos são passados a implementação do objeto. Posteriormente, no *skeleton* os resultados da operação são serializados e enviados de volta ao *stub*, que desfaz a serialização e os retorna ao chamador.

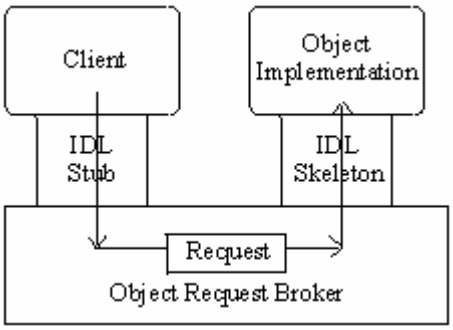


Figura V.2 Fluxo de requisição no ambiente CORBA [RV-2]

Anexo V.2 O Modelo CORBA de Segurança

O modelo CORBASec [RV-4] é uma especificação aberta para segurança em sistema de objetos distribuídos, que atualmente encontra-se na versão 1.7. Este modelo estabelece um conjunto de procedimentos para tratar da identificação e autenticação, da autorização e controle de acesso na invocação de métodos remotos, da segurança na comunicação entre objetos e outros aspectos como esquemas de delegação de direitos, não-repúdio, auditoria e administração da segurança.

O objetivo principal do CORBASec é fornecer segurança no ambiente do *ORB* na forma de objetos de serviço *COSS* (*Common Object Services Specification*). Uma especificação de objeto de serviço usualmente consiste de um conjunto de interfaces e de uma descrição do comportamento do serviço, que podem ser utilizados por objetos de

aplicação e outros objetos de serviço. A sintaxe usada para especificar as interfaces é a *OMG IDL* [RV-3]. A semântica que especifica o comportamento do serviço é expressa em termos de objetos, operações e tipos de dados padronizados.

A arquitetura CORBASec concretiza as propriedades de segurança: confidencialidade, disponibilidade, integridade, autenticidade e contabilização através de seu projeto comprometido com os princípios gerais de segurança para aplicações distribuídas da Tabela V.1

Princípio	**Descrição**
Transparência	O CORBASec interferirá o mínimo possível nas atividades do usuário.
Independência de mecanismo	O CORBASec será independente da tecnologia de segurança subjacente.
Flexibilidade	O CORBASec suportará uma variedade de políticas e mecanismos de segurança.
Interoperabilidade	O CORBASec suportará a interoperabilidade entre implementações diferentes, entre *ORB* diferentes, entre sistemas com e sem segurança, entre tecnologias de segurança diferentes e, entre diferentes domínios de um sistema distribuído – podendo suportar também diferentes políticas de segurança.
Escalabilidade	O sistema seguro se adaptará a ambos os sistemas, de pequena e larga escala, isto é, fornecerá suporte ao agrupamento de sujeitos e objetos usando grupos, *roles* e domínios de políticas gerenciáveis que poderão interoperar.
Simplicidade	O sistema de segurança será fácil de entender, usar e administrar.
Desempenho	A segurança não comprometerá o desempenho do sistema.
Certificação	Será possível avaliar e certificar a segurança do sistema e dos mecanismos subjacentes de acordo com critérios de avaliação de segurança padronizados (por exemplo, *Critérios Comuns*, ISO 15408).

Tabela V.1 – Princípios adotados no CORBASec

Uma arquitetura de segurança deve confinar a funcionalidade chave de segurança num núcleo confiável (“núcleo de segurança”), que garante a aplicação da parte essencial da política de segurança definida. Os *ORB*s seguros não necessitam suportar todas as funcionalidades[35] de segurança definidas na especificação do CORBASec.

O modelo CORBASec pode ser encarado sob o enfoque da administração – compreendendo o uso de interfaces para definir e gerenciar as políticas de segurança e os

[35] A especificação do CORBASec define 2 níveis de funcionalidades em que os *ORB*s podem se enquadrar:

- O **nível 1** (*SecuryLevel 1*) fornece segurança nas invocações entre cliente e objeto para aplicações que não têm ciência da segurança, garantindo o controle de acesso aplicado pelo ORB e auditoria de eventos do sistema. A funcionalidade de não-repudio é opcional.
- O **nível 2** (*SecuryLevel 2*), suporta as funcionalidades do nível 1 oferecendo outras interfaces de aplicação e de administração. Neste nível a aplicação pode definir suas próprias políticas de autorização através de facilidades para administração da política de segurança. Opcionalmente, a funcionalidade de não-repudio e interoperabilidade segura podem ser oferecidas neste nível.

domínios da aplicação e, da implementação – consistindo do uso das interfaces disponíveis para o implementador de objetos seguros.

Anexo V.2.1 O modelo Estrutural do CORBASec

O modelo CORBASec relaciona objetos e componentes de quatro níveis na seguinte estratificação de sistema (Figura V.3): nível de aplicação – constituindo objetos de aplicação; nível de *middleware* – composto por: objetos de serviço, serviços *ORB* e núcleo do *ORB*; nível de tecnologia de segurança – compreendendo os serviços de segurança adjacentes; e nível de proteção básica – composto por uma combinação de funcionalidades de sistemas operacionais e hardware. A seguir cada nível será comentado com maior detalhamento.

Anexo V.2.1.1 Nível de aplicação

Na Figura VIII.3, os objetos cliente (*client object*) e destino (*target object*) representam o nível de aplicação. Muitos componentes de aplicação não têm conhecimento da segurança envolvida nas invocações de objetos e precisam do *ORB* para chamar de maneira transparente os serviços de segurança necessários para tal operação.

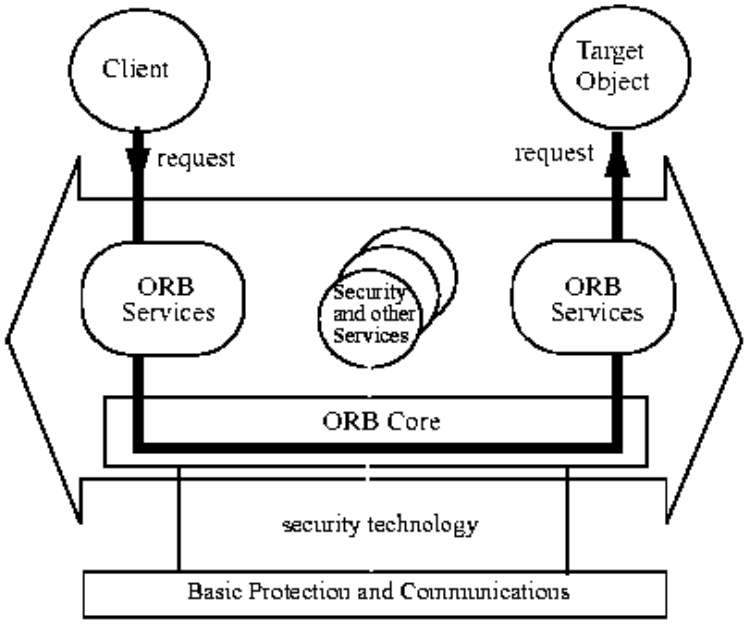


Figura V.3 – Modelo estrutural do CORBASec [RV-4]

Porém, outras aplicações têm capacidade de garantir sua própria segurança através da ativação direta dos serviços de segurança. Assim, o CORBASec suporta ambas as situações através de suas funcionalidades de implementação de segurança.

Anexo V.2.1.2 Nível de middleware

O núcleo do *ORB* (*ORB core*) suporta a funcionalidade mínima para habilitar um cliente a invocar uma operação num objeto de destino. Os serviços *ORB* e os objetos de serviço (*security and other services*) *COSS* de segurança da Figura V.3 são construídos sobre o núcleo *ORB* e estendem as funções básicas de comunicação com características ou controles adicionais, facilitando a implementação de objetos distribuídos. Uma combinação de serviços *ORB* e serviços *COSS* são utilizados na implementação da segurança no modelo CORBA.

Nas especificações do modelo CORBASec, os chamados serviços *ORB* são constituídos pelos *interceptors* – interpostos no caminho de uma chamada do cliente para um objeto. Cada serviço *COSS* relacionado com a segurança é associado a um *interceptor*, cuja finalidade é desviar de modo transparente a invocação de um método para ativar o objeto de serviço associado.

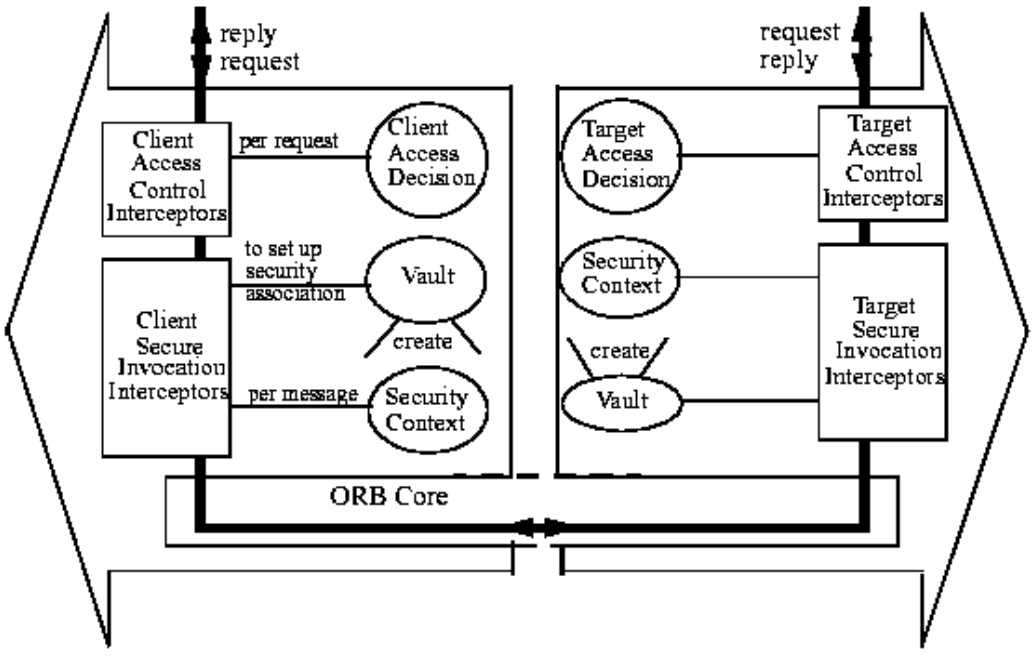


Figura V.4 – *Interceptors* do CORBASec [RV-4]

No modelo CORBASec são definidos dois *interceptors* (Figura V.4) que atuam durante uma invocação de método: o *Access Control Interceptor* – provoca um desvio de nível mais alto para a realização do controle de acesso na chamada, e o *Secure Invocation Interceptor* – realiza uma interceptação de nível mais baixo no sentido de fornecer propriedades de integridade e de confidencialidade nas transferências de mensagens correspondentes à invocação. Esses interceptadores atuam tanto no lado cliente da invocação como no lado do objeto servidor da aplicação.

Anexo V.2.1.3 Tecnologia de segurança

Como visto na Figura V.3 a camada do *ORB* é isolada tanto da aplicação quanto da tecnologia de segurança (*security technology*) através dos objetos de serviços do CORBASec. Na tecnologia de segurança são implementadas várias funcionalidades de objetos de serviços CORBA relacionados a segurança, dentre estas podem estar incluídos serviços adjacentes de autenticação, de manipulação de certificação e de associação segura. Várias tecnologias podem oferecer estes serviços e, além disto, é possível utilizar implementações de interfaces genéricas, tais como *GSS-API* (*Generic Security Services API – RFC 2743*) – que encapsulam os detalhes dos mecanismos de segurança subjacentes para os serviços de segurança CORBA.

Anexo V.2.1.4 Proteção básica e comunicação

A proteção básica (Figura V.3 – *basic protection and communications*) diz respeito ao fornecimento de meios para proteger componentes da aplicação uns dos outros e os componentes de suporte dos serviços de segurança no ambiente operacional.

Se o ambiente operacional – incluindo o sistema operacional e o suporte para a comunicação remota, possui recursos para o transporte seguro de mensagem, não será necessária a manipulação criptográfica das mensagens em nível de *ORB.*

Em geral, deve-se estabelecer limites de proteção em torno de grupos de componentes pertencentes a um domínio de proteção. Componentes constituindo um domínio de proteção (mesmo o sistema operacional e o *hardware*) não possuem suas

interações mediadas pelos serviços de segurança CORBA; estas interações são controladas através do uso de mecanismos do sistema operacional.

Anexo V.2.2 O controle de acesso no CORBASec

O CORBASec permite definir políticas de autorização usando mecanismos de *ACL*s, listas de *capabilities* e controle de acesso baseado em *roles* (*RBAC*). A especificação provê suporte direto a políticas de autorização discricionárias através dos objetos *DomainAccessPolicy* e *RequieredRights* (Figura V.5) e usa *ACL*s para armazenar os direitos de acesso. A especificação não prevê suporte direto ao uso de *capabilities* e ao uso de *RBAC* – oferecido de maneira simplificada através dos atributos de privilégio do tipo *role*.

Os controles de autorização no CORBASec permitem que as políticas de autorização sejam verificadas em dois níveis, *middleware* e aplicação (Figura V.5).

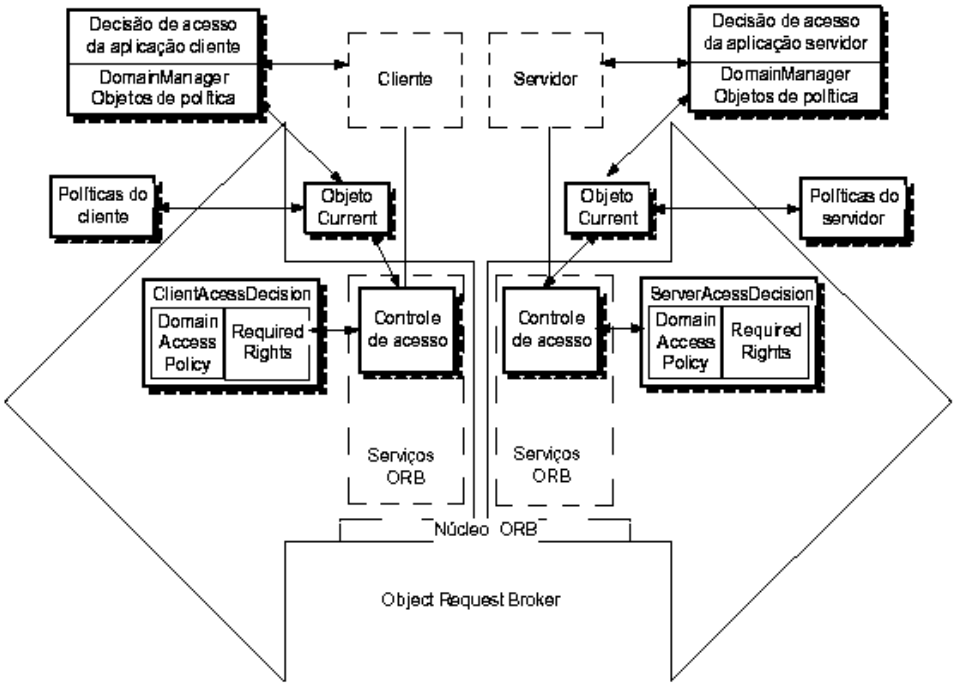


Figura V.5 – Modelo de Controle de acesso do CORBASec [RV-4]

Em nível de *middleware* a verificação é feita utilizando objetos de serviço durante uma invocação de método (*object invocation access policy*) e, portanto, de forma

transparente para a aplicação (*security unaware*). Neste nível, as políticas de segurança são descritas na forma de atributos de segurança dos recursos do sistema (atributos de controle) e dos principais (atributos de privilégio). O objeto *DomainAccessPolicy* representa a interface de acesso à política de autorização discricionária, representando para um conjunto de principais o montante de direitos na execução de operações em todos os objetos de um sistema.

Em nível de aplicação as políticas de acesso são específicas da aplicação (*application object access policy*) que precisa definir funções para impor o cumprimento das suas próprias regras de controle de acesso (*security aware*). Neste caso a especificação não possui interfaces administrativas para este tipo de política.

Anexo V.3 Considerações finais

Neste anexo foi introduzido de maneira breve e geral, principalmente, o modelo de segurança da arquitetura CORBA, sem considerar aspectos muito específicos da especificação.

Anexo V.4 Referências bibliográficas

[RV-1] - *Object Management Group*. *CORBA 2.6 Specification*, http://www.omg.org/cgi-bin/doc?formal/01-12-01.

[RV-2] – Vinoski, Steve. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, February, 1997.

[RV-3] - *Object Management Group. CORBA 2.3 OMG IDL*, http://www.omg.org/cgi-bin/doc?formal/00-04-01.

[RV-4] - *Object Management Group. Security Service, v1.7*, http://www.omg.org/cgi-bin/doc?formal/2001-03-08.

Anexo VI. Representação verificação da política de autorização focada na especificação

Anexo VI.1 Considerações iniciais

Para [RVI-1] a concessão ou negação de direitos de acesso deve ser explícita para evitar contradições na representação (*representation*) e na verificação (*evaluation*) das regras da (política de) autorização. Além disto, as regras de autorização devem ser expressas utilizando uma especificação formal, sem compromisso com o mecanismo de implementação da autorização, para não ficar limitado aos recursos que o mesmo oferece para tal. Assim, o autor define um modelo onde um sujeito $\underline{s}$ para ter acesso a um objeto $\underline{o}$, deve ter direitos de acesso $\underline{r}$ e enviar ao módulo de autorização uma requisição com o seguinte formato, *req* ($\underline{r}$, $\underline{s}$, $\underline{o}$). O módulo de autorização, como resultado do processo de verificação da autorização poderá emitir um dos seguintes pareceres:

a) *grant* ($\underline{r}$, $\underline{s}$, $\underline{o}$) $\rightarrow$ se e somente se ($\underline{r}$, $\underline{s}$, $\underline{o}$) $\in P^+$.

b) *deny* ($\underline{r}$, $\underline{s}$, $\underline{o}$) $\rightarrow$ se e somente se ($\underline{r}$, $\underline{s}$, $\underline{o}$) $\in N^+$.

c) *fail* ($\underline{r}$, $\underline{s}$, $\underline{o}$) $\rightarrow$ se e somente se ($\underline{r}$, $\underline{s}$, $\underline{o}$) $\notin P^+ \cup N^+$.

Neste caso, P^+ representa os direitos explicitamente concedidos (*grant*), N^+ os direitos explicitamente negados (*deny*) e uma resposta *fail* significa que a especificação está incompleta ou incorreta.

Além da dinâmica do sistema, o autor apresenta uma série de teoremas, provas e corolários que definem a sintaxe e a semântica básicas para especificação das regras (da política) de autorização de uma maneira “modular” e “procedural” (como numa linguagem

de programação “C” ou pascal, por exemplo) e não apenas declarativa com é feito na representação clássica de autorização usando *ACL* .

A seguir será introduzida uma noção do framework que se utiliza das idéias expostas acima para representar e interpretar a política de segurança.

Anexo VI.2 Framework para autorização distribuída

No *framework* [RVI-2] é proposta uma linguagem, *Generalized Access Control List* (*GACL*), para descrição dos atributos (*requirements*) de autorização [RVI-3]. Para expressar a (política de) autorização a *GACL* usa em cada uma de suas entradas a seguinte composição: um rótulo com o nome do objeto, a palavra *declare* (declaração) e uma lista (*list*) de direitos.

Após a palavra *declare* é possível especificar:

- *ordered* → indicando que a parte *list* deve ser interpretada seqüencialmente. Se nada for especificado, a parte *list* é avaliada *unordered*, ou seja, todas as entradas devem ser verificadas para que a decisão de permitir (*grant*), negar (*deny*) ou falhar (*fail*) seja tomada pelo módulo de autorização.
- *anonymous* → indicando que o servidor de aplicação aceita autorizações certificadas por servidores de autorização mesmo sem saber quem é o cliente.

Na parte *list* são especificadas as entradas que definem os direitos de acesso ao objeto, neste contexto serão especificados apenas alguns dos parâmetros possíveis:

- A:: → indica que se um sujeito tinha um determinado direito em *A* o terá onde esta linha for especificada.
- *default* → indica a política de autorização global. Por exemplo, *default::<[*],[-*]>*, indica que todos (*[*]*) os sujeitos têm os direitos negados (*[-*]*).
- *Inherit* → herda os direitos por exigência ou necessidade (*demand*), ou sempre (*always*).

Para ilustrar como a *GACL* é verificada, considere os usuários *Alice* e *Bob* e os grupos *Dept* e *Research*, que têm os seus *requiremets* de autorização expressos conforme visto na representação parcial da *GACL* mostrada na Figura VI.1 – para os objeto *P.exe* e *P.src*. Como a verificação começa em *P.exe*, vê-se após o *declare* que há a palavra *ordered*, indicando que a verificação da parte *list* deve ser feita seqüencialmente. Assim, na

assertiva *‘<[Alice,Bob],[-execute]’>’* é especificado que os usuários *Alice* e *Bob* não têm permissão de execução no objeto *P.exe*, ao contrário do especificado na assertiva seguinte, ‘<[*Dept*],[*execute]>’*, que permite a execução aos membros do grupo *Dep*. Já a assertiva ‘*inherit P.src::<[*],[write*]>’ especifica que todos os usuários que têm permissão de escrita em *P.src* também a terão em *P.exe*. No caso, apenas os membros do grupo *Research*.

P.exe	*declare*	*Ordered*
	list	<[Alice,Bob],[*-execute*]>,
		<[*Dept*],[*execute*]>,
		inherit P.src::<[*],[*write*]>
P.src	*declare*	
	list	<[Research],[*read,write*]>,
		<[*-Dept*],[*-write*]>

Figura VI.1 - Representação da política de autorização utilizando a GACL

O *framework* também suporta delegação e sugere uma infra-estrutura de servidores para autorização distribuída incluindo operações e protocolos, que não serão abordados no contexto deste anexo.

Anexo VI.3 Considerações finais

Neste anexo foram mostrados sucintamente alguns aspectos relativos a representação (de uma política) de autorização sem compromisso com o mecanismo que a implementa, expressa através da generalização da *ACL* clássica.

Anexo VI.4 Referências bibliográficas

[RVI-1] – Woo, T. Y.C. e Lam, S.S. *Authorization in Distributed Systems: A Formal Approch*, *Proceedings of IEEE symposium on R. in Security and Privacy*, 1992.

[RVI-2] – Woo, T. Y.C. e Lam, S.S. *A Framework for Distributed Authorization*, *Proceedings ACM conference on Computer and Communications Security*, 1993.

[RVI-3] - T. Y. C. Woo e S. S. Lam. *Designing a distributed authorization service*, *Proceedings IEEE INFOCOM '98*, 1998.

Anexo VII. Representação e verificação da política de autorização utilizando mecanismos estendidos de autorização

Anexo VII.1 Considerações iniciais

Para [RVII-1] pode-se pensar a (política de) autorização associada a um recurso protegido como um conjunto de operações que são permitidas a um grupo definido de principais em um dado recurso com restrições de concessão (opcionais). Para suportar tais restrições condicionais nos direitos de acesso [RVII-2] a abstração de *ACL* e *capability* deve ser estendida. Assim, a (política de) autorização pode ser descrita usando um mecanismo de *EACL* (*Extend Access Control List*) ou "*Extend" capability*, onde cada entrada do mecanismo utilizado descreve para cada recurso uma lista de entidades válidas para cada fim, o conjunto de direitos de acesso concedidos e as respectivas restrições de concessão.

A seguir são apresentados os aspectos mais relevantes do *framework* concebido baseado nas idéias citadas acima.

Anexo VII.2 Framework de autorização para sistemas distribuídos

No *framework* [RVII-1] [RVII-2] a linguagem para representar a (política - *policy language* de) autorização é uma seqüência de *tokens*. Cada *token* consiste de: um tipo (*token type)* - *tokens* do mesmo tipo têm a mesma semântica de autorização; uma autoridade de definição (*defining authority*) - responsável por definir o valor dentro do tipo de atributo; e um valor (*value*) - valor do atributo de autorização.

Na implementação da (política de) autorização é possível definir: a identificação do principal autorizado ao acesso, a identificação do outorgante (*grantor*), um conjunto de direitos de acessos e um conjunto de condições.

° Identificação do principal: representa a identificação a ser utilizada para o propósito de controle de acesso e pode ser: usuário, *host*, aplicação, *CA*, grupo ou *anybody*.

° Identificação do outorgante (*grantor*): representa a identificação utilizada para especificar o outorgante de uma *capability* ou de uma credencial delegada.

° Direitos de acesso: indicam de maneira explicita qual *principal* ou grupo tem autorização permitida ou negada para executar alguma operação.

° Condições: podem ser genéricas ou específicas.

- Condições específicas: se as condições gerais não forem suficientes para especificar as condições desejadas, é possível especificá-las usando *tokens* cuja interpretação será de responsabilidade da aplicação.

- Condições genéricas: são as condições que podem ser avaliadas pelo mecanismo de controle de acesso. As condições genéricas previstas pelo *framework* podem ser vistas na tabela [Tabela VII.1]:

horário (*time*)	indica o período de tempo em que o acesso é concedido.
localização(*location*)	indica a localização do *host* ou domínio que tem autorização concedida.
conexão (*connection*)	indica informações de segurança da conexão (com integridade?, com confidencialidade?, rede particular? ou *dialup*?, ...).
restrições de privilégio (*privilege constraints*)	indica restrições para suportar separação de tarefas.
restrições de segurança multinível (*multi-level security constraints*)	indica restrições de confidencialidade e integridade específicas de políticas obrigatórias (mandatórias).
pagamento (*payment*)	especifica a “moeda corrente” e o montante que deve ser “pago” antes de acessar um objeto.
cota (*quota*)	especifica a “moeda corrente” e o limite (quantidade de recurso que pode ser consumido ou obtido).
força da autenticação (*strength of authentication*)	especifica o mecanismo de autenticação ou um conjunto de mecanismos para autenticar o usuário.
atributos do sujeito (*attributes of subject*)	especifica os atributos, por exemplo *clearence*, que o sujeito deve possuir para acessar um objeto.
restrições de confiança (*trust constraints*)	representa restrições colocadas na credencial de segurança.

Tabela VII.1- Condições genéricas prevista no *framework* de autorização

A Figura VII.1 mostra um exemplo de *EACL* implementado *tokens* especificando que o usuário *Tom* identificado numa credencial *X.509* (#1) e o grupo *admin* do domínio *kerberos* ORG.EDU (#2) estão autorizados (*pos_access_rights*) a ter acesso '*read'* e '*read* e *write'*, respectivamente, ao objeto *FILE* que é responsabilidade do gerenciador de arquivos (*local_manager*).

EACL entry	*token type*	*def.authority*	*value*
#1	access_id_USER	X.509	"/C=us/O=alliance/CN=Tom"
	pos_access_rights	local_manager	FILE:read

EACL entry	*token type*	*def.authority*	*value*
#2	access_id_GROUP	kerberos.V5	admin@ORG.EDU
	pos_access_rights	local_manager	FILE:read,write

Figura VII.1 – Exemplo de *EACL*, implementando *tokens*

A verificação da autorização é feita através da *Generic Authorization e Access API* que oferece um conjunto de funções com interfaces padronizadas. Estas APIs devem ser implementadas junto com o mecanismo de controle de acesso. Maiores detalhes podem ser obtidos em http://www.isi.edu/gost/info/gaaapi/doc/refman.pdf.

Anexo VII.3 Considerações finais

Neste anexo, foi introduzida uma linguagem de política baseada nos recursos que os mecanismos de segurança oferecem. Porém, esta linguagem é genérica o suficiente para permitir a especificação dos atributos de autorização suportados por qualquer um dos mecanismos estendidos de autorização (*EACL* ou *capability*). Foi comentado também, mas não detalhado, que existe uma *API* (*GAA*) para fazer a verificação da política especificada com esta linguagem de política de autorização. Salientando ainda, que em relação a anterior (anexo VI), esta linguagem de política tem a vantagem de suportar o mecanismo de *capability* e não apenas uma "variação de *ACL*" – *GACL*, como a anterior. O *framework* apresentado neste anexo se encontra em processo de padronização no *IETF* (*Internet Engineering Task Force*) [RVII-3].

Anexo VII.4 Referências bibliográficas

[RVII-1] – Ryutov, Tatyana and Neuman B. Clifford. *Representation and Evaluation of Security Policies for Distributed System Services*. *Proceedings of the* DISCEX, Hilton Head Island, SC, Janeiro 2000.

[RVII-2] - Gheorghiu, G., Ryutov, T., Neuman, C. *Authorization for Metacomputing applications, Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, Chicago, Illinois, Julho de 1998.

[RVII-3] - *The Internet Engineering Task Force - CAT Working Group, Access Control Framework for Distributed Applications*. http://www.isi.edu/gost/info/gaaapi/doc/drafts/frmw_draft5.txt, 2000.

Referencias Bibliográficas

[1] Denning, Dorothy. *Criptography and data security*. Addison-Wesley. 1982.

[2] C. E. Landwehr. *omputer security*. IJIS (2001) Vol.1, p.p. 3–13. 2001.

[3] Shirley, R. *Internet Security Glossary*. IETF, RFC 2828. 2000.

[4] Mackenzie, D. E Pottinger, G. *Mathematics, Technology, and Trust: Formal Verification, Computer Security, and the U.S. Military*, IEEE Annals of the History of Computing, Vol. 19, Nº 3.1997.

[5] Goguen, J. A. and Mesajuer, J. *Security Policies And Security Models*. Proceedings of IEEE symposium on Reseach in Security and Privacy. 1982.

[6] Sandhu, R. and Samarati, P. *Authentication, Access Control, and Audit*. ACM Computing Surveys, Vol. 28, No. 1. 1996.

[7] Osborn, S., Sandhu, R. and Munawer, Q. *Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies*. ACM Transactions on Information and System Security, Vol. 3, No. 2, pag. 85–106. 2000.

[8] Lampson, B. W. *Protection*. Proc. 5th Princeton Conf. on Information Sciences and Systems, Princeton, p. 437. 1971.

[9] Landwehr, C. E., Heitmeyer, C.L. and Mclean, J. *A security Model for Military Message Systems*. ACM Transactions on Computer Systems, Vol. 2, Nº 3, pg. 198-222. 1984.

[10] Bell, D. Elliott; La Padula, L. J. *Secure Computer Systems : Mathematical Foundations*. MTR-2547 Vol. I, The MITRE Corporation, Bedford, Massachusetts. 1973.

[11] McLean, J. *The Specification and Modeling of Computer Security*. IEEE Computer, Vol. 23, Nº 1.1990.

[12] Clark, D. D. and Wilson, D. I. *A Comparison of Commercial and Military computer Security Policies*. Proceedings of IEEE Symposium on Security and Privacy. 1987.

[13] Denning, D. *A Lattice Model of Secure Information Flow*. Communication of ACM, Vol. 19, No. 5. 1976.

[14] Sandhu, R., Ferraiolo, D. and Kuhn, R. *The NIST Model for Role-Based Access Control: Towards A Unified Standard*. Proceedings of the fifth ACM workshop on Role-based access control. 2000.

[15] Westphall, C. M. *Um esquema de autorização para a segurança em sistemas distribuídos de larga escala*. Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina, Tese de doutorado, Florianópolis – Santa Catarina. 2000.

[16] Anderson, J. P. *Computer Security Technology Planning Study*. Report ESD-TR-73-51.Electronic Systems Division. 1972.

[17] Landwehr, C. E. *Best available technologies for computer security*. IEEE Computer Vol. 16, No. 7, pág. 86-100. 1983.

[18] Landwehr, C. E., Heitmeyer, C.L. and Mclean, J. *A security Model for Military Message Systems*. ACM Transactions on Computer Systems, Vol. 2, Nº 3, pg. 198-222. 1984.

[19] Glenn, F. *RBAC in UNIX Administration*. Proceedings of the fourth ACM workshop on role-based access control, pg. 95 –101. 1999.

[20] *The Internet Engineering Task Force*. *Kerberos Network Authentication Service (Versão 5)*. [online] http://www.ietf.org/rfc/rfc1510.txt. 1993.

[21] International Telecommunication Union. *Telecommunication Standardization Sector - Publications*. Recommendation X.509 – Information technology – Open Systems Interconnection – The Directory Authentication Framework. 1993.

[22] Abadi, M., Burrows, M., and Lampson, B. *A Calculus for Access Control in Distributed Systems*. ACM Transaction on Programming Language and Systems, Vol. 15, Nº 4, pg. 706-734.1993.

[23] Gasser, M., Goldstein, A., Kaufmann, C. and Lampson, B. *The Digital Distributed Security Architecture*. Proceedings of National Computer Security Conference. 1989.

[24] Linn, J. *Practical Authentication For Distributed Computing*. Proceedings of IEEE symposium on Security and Privacy. 1990.

[25] Lampson, B. Abadi, M. Burrows, M. and Wobber, E. *Authentication in Distributed System:Theory and Pratice*, ACM transaction on Computer Systems, V. 10, Nº 4, pg. 265-310. 1992.

[26] Woo, T. Y.C. and Lam, S.S. *Authentication for distributed System*. Computer, V. 25, Nº 1, pg. 39-52. 1992.

[27] Gasser, M., and McDermott, E. *An Architecture for Practical Delegation in a Distributed System*. Proceedings of IEEE symposium on Security and Privacy. 1990.

[28] Woo, T. Y.C. and Lam, S.S. *A Framework for Distributed Authorization*. Extended Abstract, Proceedings ACM conference on Computer and Communications Security. 1993.

[29] Ryutov, T. and Neuman, B. C. *Representation and Evaluation of Security Policies for Distributed System Services*. Proceedings of the DISCEX, Hilton Head Island, SC. 2000.

[30] Woo, T. Y.C. and Lam, S.S. *Authorization in Distributed Systems: A Formal Approch*. Proceedings of IEEE symposium on Reseach in Security and Privacy. 1992.

[31] Departament of Defense. *Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD. 1985.

[32] RSA Laboratories. *PKCS Standard series (#1 à #10)*. [online] http://www.rsasecurity.com/rsalabs/pkcs/index.html. 1993.

[33] National Institute of Standards and Technology. *Minimum Interoperability Specifications for PKI Components, Version 1*. [online] http://csrc.ncsl.nist.gov/pki/documents/mispcv1.ps. 1997.

[34] Powell, D. *Delta-4 Overall System Specification*. An publication: The Delta-4 Project Consortium, Echirolles, France, ISBN:2-907801-00-7. 1988.

[35] Fraga, J. S. *La Sécurité des Donnés par la Tolérance aux Instrusions*. PhD thesis, Institut National Polytechnique de Toulouse. 1985.

[36] National Institute of Standards and Technology (NIST). *Report of Federal Bridge Certification Authority Initiative and Demonstration*. [online] Disponível em

http://csrc.nist.gov/pki/documents/emareport_20001015.pdf. Acesso em junho de 2003.

[37] Garfinkel, S. *PGP:Pretty Good Privacy*. O'Reilly & Associates, Inc. 1995.

[38] Blaze M., Feigenbaum J. and Lacy J. *Decentralized Trust Management*, Proceedings of the 17th IEEE Symp. on Security and Privacy, IEEE Computer Society. 1996.

[39] Blaze, M., Feigenbaum, and Strauss M. *Compliance-Checking in the PolicyMaker Trust-Management System.* Proc. 2nd Conference on Financial Cryptography. 1998.

[40] The Internet Engineering Task Force. *The KeyNote Trust-Management System*. [online] http://www.ietf.org/rfc/rfc2704.txt. 1999.

[41] The Internet Engineering Task Force. *SPKI Requirements*. [online] http://www.ietf.org/rfc/rfc2692.txt. 1999.

[42] The Internet Engineering Task Force. *SPKI Certificate Theory*. [online] http://www.ietf.org/rfc/rfc2693.txt. 1999.

[43] Lampson, B. e Rivest, R. L. *A simple Distributed Security Infrastructure*. [online] http://theory.lcs.mit.edu/~cis/sdsi.html.1996.

[44] Abadi, M. *On SDSI's Linked Local Name Spaces*. Proceedings of of The 10th Computer Security Foundations Workshop. 1997.

[45] Rivest, R. L. *S-expressions*. [online] http://theory.lcs.mit.edu/ /~rivest/sexp.html. 1997.

[46] Nikander, P. and Viljanen, L. *Storing anda Retrieving Internet Certificates*. The Third Nordic Workshop on Secure IT Systems, Trondhein, Norway. 1998.

[47] Aura, T. *Fast Access Control Decisions from Delegation Certificate Databases*. proceedings of 3th Australasian Conference on Information Security and Privacy, Berlin. 1998.

[48] Ajmani, S. A T*rusted Execution Platform for Multiparty Computation*, master thesis, Department of Eletrical Engeneering and Computer Science, MIT. 2000.

[49] Li, N. *Local Names in SPKI/SDSI*. In: proceedings of the IEEE Computer Security Foundations Workshop, Los Alamitos, CA. 2000.

[50] Clarke, D. E. *SPKI/SDSI HTTP Server Certificate Chain Discovery in SPKI/SDSI*. Department of Eletrical Engeneering and Computer Science of MIT, master dissertation. 2001.

[51] Santin, A. O., Fraga, J. S., Mello, E. R., e Siqueira, F. *Teias de Federações como Extensões ao Modelo de Autenticação e Autorização SDSI/SPKI*. In: XXI SBRC`2003, Natal - RN. 2003.

[52] Aura, T. *On the structure of delegation networks*, proceedings of 11th IEEE Computer Security Foundations Workshop, Rockport, MA USA.1998.

[53] Brewer, D. and Nash, M. *The chinese Wall Security Policy*. Proceedings of IEEE Symp. on Security and Privacy, IEEE Computer Society, pág. 206-214. 1989.

[54] Elien, J. E. *Certicate discovery using SPKI/SDSI 2.0 certicates*. master dissertation, MIT. 1998.

[55] Berners-Lee, T., Fielding, R. and Masinter, L. *Uniform Resource Identifiers (URI): Generic Syntax*, IETF RFC1396. 1998.

[56] Santin, A. O., Fraga, J. S., Mello, E. R. e Siqueira, F. *Federation Web: A Scheme to Compound Authorization Chains*. In: The 22nd Symposium on Reliable Distributed Systems, 22nd IEEE-SRDS´2003, Florence, IT. 2003.

[57] Loerch, U. *An Introduction to Graph Algorithms*. Department of Computer Science. Available in http://www.cs.auckland.ac.nz/~ute/220ft/graphalg/node8.html. 2000.

[58] Santin, A. O., Fraga, J. S. and Maziero, C. *Extending the SDSI/SPKI model through federation webs*. In: Seventh IFIP TC-6 TC-11 CMS`2003, LNCS 2828, Torino, IT. 2003.

[59] C. Adams. *The Simple Public-Key GSS-API Mechanism* (SPKM). IETF RFC 2025. 1996.

[60] Parker, T. and Pinkas, D. *SESAME v4 - Overview*. [online] https://www.cosic.esat.kuleuven.ac.be/sesame/doc-ps/overview.ps, 1995. Visitado em 03/02/2003.

[61] Hartman, B., Flinn, D. J. and Beznosov, K. *Enterprise Security with EJB and CORBA*. John Wiley & Sons. ISBN 0-471-40131-5. 2001.

[62] Object Management Group (OMG). *Resource access decision facility specification*. OMG Document 01-04-01. 2001.

[63] Abts, C. M. *COTS Software Integration Cost Modeling Study*. Center for Software Engineering – University of southern California. 1997. [online] http://sunset.usc.edu/research/COCOTS/docs/USAFReport.pdf.

[64] Adiron, LLC. *ORBAsec SL2 User Guide*. Version 2.1.4. 2000.

[65] Iona Technologies, Inc. *Orbacus User Guide*. Version 3.3.4. 2001.

[66] IAIK. *iSaSiLk 3* - Reference Manual. Institute for Applied Information Processing and Communications (IAIK). Version 3. 2000.

[67] Morcos, A. *A Java implementation of Simple Distributed Security Infrastructure*. master dissertation, MIT. 1998.

[68] Mello, E. R. *Redes de Confiança em Sistemas de Objetos CORBA*. Universidade Federal de Santa Catarina – Departamento de Automação e Sistemas, dissertação de mestrado, [online] disponível em: http://www.das.ufsc.br/~emerson/files/mello2003-dissertacao.pdf. 2003.

[69] Object Management Group (OMG). *Security Service, v1.7*. [online] http://www.omg.org/cgi-bin/doc?formal/2001-03-08. 2001.

[70] National Institute of Standards and Technology (NIST). *Entity Authentication Using Public Key Cryptography*. 1997. FIPS PUB 196. [online] Disponível em http://csrc.nist.gov/publications/fips/fips196/fips196.pdf. Acesso em junho de 2003.

[71] Mello, E. R., Fraga, J. S., Santin, A. O. and Siqueira, F. *Redes de Confiança em Sistemas de Objetos CORBA.* 5º SSI`2003 - ITA - São José dos Campos – SP. 2003.

[72] Object Management Group (OMG). *The Common Object Request Broker Architecture v2.6.* OMG Document 01-12-30. 2001.

[73] Bray, T., Paoli, J., and Sperberg-McQueen, C. M. *Extensible Markup Language (XML) 1.0 - W3C recommendation*. 1998. Relatório Técnico REC-xml-19980210, W3C. http://citeseer.nj.nec.com/bray98extensible.html. Visitado em 04/02/2003.

[74] Mello, E. R., Boesel, D. F. e Carrijo, L. F. *Biblioteca parserSxxS*. Relatório interno DAS/cadeias de confiança. 2003.

[75] Orri, X. and Mas, J.-M. *SPKI-XML Certificate Structure*. 2001. Internet Engineering Task Force - Internet Draft. http://xml.coverpages.org/ni2001-12-18-a.html. Visitado em 02/04/2003.

[76] Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Hors, A. L., Nicol, G., Robie, J., Sutor, R., Wilson, C., and Wood, L. *Document Object Model Level 1 Specification version 1.0* - W3C recommendation, 1998. Relatório técnico, W3C. http://www.w3.org/TR/REC-DOM-Level-1/. Visitado em 04/02/2003.

[77] Megginson, D. *The Simple API for XML*. 1998. [online] http://www.saxproject.org/. Última visita em Janeiro de 2003

[78] Staken, K. *Developers Guide 0.7.1*. 2002. [online] http://xml.apache.org/xindice/guide-developer.html, última visita em Janeiro de 2003.

[79] Clark, J. and DeRose, S. *XML Path Language (XPath) Version 1.0* - W3C recommendation. 1999. Techical report REC-xpath-19991116, W3C. http://www.w3.org/TR/xpath.Visitado em 04/02/2003.

[80] Lampinen, T. *Using SPKI certificates for authorization in CORBA based distributed object-oriented systems*. Proceedings of the 4th Nordic Workshop on Secure IT Systems (NordSec'99), Kista, Sweden. 1999.

[81] Ellison, C. and Schneier, B. *Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure*. Computer Security Journal, v 16, n 1, pp. 1-7. 2000. [online] Disponível em http://www.counterpane.com/pki-risks.html. Acesso em junho de 2003.

[82] Common Criteria Implementation Board (ISO). *Common Criteria for Information Security Evaluation*. ISO IS 15408. 1999.

[83] Stoneburner, G., Goguen, A.and Feringa, A. *Risk Management Guide for Information Technology Systems*, 2001. NIST Special Publication 800-30. [online] Disponível em http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf. Acesso em junho de 2003.

[84] Object Management Group (OMG). *Security Service Specification: Appendix D.2 - Protecting Against Threats*, pg. 369-374. 2002. [online] Disponível em http://www.omg.org/technology/documents/formal/security_service.htm. Acesso em junho de 2003.

[85] Martin, R. A.. *Managing Vulnerabilities in Networked Systems*. IEEE Computer Society magazine, Computer, Vol. 34, No. 11. 2001.

[86] The MITRE Corporation. *Common Vulnerabilities and Exposures*. [online] Disponível em http://cve.mitre.org/. Acesso em junho de 2003.

[87] Kaksonen, R. *A Functional Method for Assessing Protocol Implementation Security*. In: VTT publications 448, Technical Research Centre of Finland. Parte do PROTOS project - Thesis licentiate. 2001.

[88] Mell, P. and Grance, T. *Use of the Common Vulnerabilities and Exposures (CVE) Vulnerability Naming Scheme*. 2002. NIST Special Publication 800-51. [online]

Disponível em http://csrc.nist.gov/publications/nistpubs/800-51/sp800-51.pdf. Acesso em junho de 2003.

[89] Anderson, A., Longley, D. and Kwork, L. *Security Modeling for Organization*, ACM, pg 241-249. 1994.

[90] Serrano, M. and Boehm, H. *Understanding Memory Allocation of Scheme Programs*. Proceedings of ACM ICFP'00, pág. 145-256. 2000.

[91] Paw, W. E Sevitsky G. *Visualizing Reference Patterns for Solving Memory Leaks in Java*. In: Proceedings of ECOOP'99, pág. 116-134. 1999.

[92] Quest Software. *Debugger, JProbe Memory*. [online] Disponível em http://java.quest.com/jprobe/debugger.shtml. Acesso em junho de 2003.

[93] Flanagan, C. and Freund, S. N. *Type-based race detection for java*. In Proceedings of the ACM SIGPLAN, 2000.

[94] Quest Software. Threadalyzer, JProbe. [online] Disponível em http://java.quest.com/jprobe/threadalyzer.shtml. Acesso em junho de 2003.

[95] Mello, E. R., Fraga, J. S., Santin, A. O. and Siqueira, F.. *Building Trust Chains Between CORBA Objects* In: 1st LADC . São Paulo - BR. LNCS 2847. 2003.

[96] Ptacek, T. H. and Newsham, T. N. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*. Originalmente da Secure Networks, Inc. 1998. [online] Disponível em http://www.insecure.org/stf/secnet_ids/secnet_ids.html. Acesso em junho de 2003.

[97] Menezes, A. J., Oorschot, P. C., and Vanstone, S. A.. *Handbook of Applied Cryptography*. CRC Press. Disponível em http://www.cacr.uwaterloo.ca/hac/. 1997.

[98] Carreira, J., Costa, D. and Silva, J. *Fault Injection Spot-checks Computer System Dependability*. IEEE Spectum, pág. 50-55. 1999.