CÁSSIO DITZEL KROPIWIEC

PROPOSTA DE UM MECANISMO DE SEGURANÇA PARA WEB SERVICES BASEADO EM IPSEC

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná, como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

CURITIBA
2003

CÁSSIO DITZEL KROPIWIEC

PROPOSTA DE UM MECANISMO DE SEGURANÇA PARA WEB SERVICES BASEADO EM IPSEC

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná, como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

Área de Concentração:
Metodologia e Técnicas de Computação

Orientador: Prof. Dr. Edgard Jamhour

CURITIBA
2003

Kropiwiec, Cássio Ditzel

Proposta de um Mecanismo de Segurança para Web Services Baseado em IPsec. Curitiba, 2003. 193 p.

Dissertação – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática Aplicada.

1. Segurança 2. Web Services 3. IPsec 4. WSDL I. Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática Aplicada II-t.

Pontifícia Universidade Católica do Paraná
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Informática Aplicada

ATA DA SESSÃO PÚBLICA DE DEFESA DE DISSERTAÇÃO DE MESTRADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA DA PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

DEFESA DE DISSERTAÇÃO Nº 085

Aos 28 dias do mês de agosto de 2003 realizou-se a sessão pública de defesa da dissertação **"Proposta de um Mecanismo de Segurança para Web Services Baseado em IPsec"**, apresentado por **Cássio Ditzel Kropiwiec** como requisito parcial para a obtenção do título de **Mestre em Informática Aplicada,** perante uma Banca Examinadora composta pelos seguintes membros:

Prof. Dr. Edgard Jamhour PUCPR (Orientador)	assinatura	aprovada parecer (aprov. reprov.)
Prof. Dr. Lau Cheuk Lung PUCPR		aprovado
Prof.Dr. Paulo L. de Geus UNICAMP		aprovado

Conforme as normas regimentais do PPGIA e da PUCPR, o trabalho apresentado foi considerado aprovado (*aprovado/reprovado*), segundo avaliação da maioria dos membros desta Banca Examinadora. Este resultado está condicionado ao cumprimento integral das solicitações da Banca Examinadora, conforme registrado no Livro de Defesas do programa.

Prof. Dr. Carlos Maziero
Diretor do PPGIA PUCPR 05/09/03
Data e assinatura, após homologação da defesa pelo colegiado

À minha esposa, **Edelyse**, **pelo**
apoio e compreensão **constantes**.

Agradecimentos

Ao professor Edgard Jamhour, pela sugestão do tema, constante apoio e confiança.

À minha família, que sempre acreditou no sucesso deste trabalho.

Ao Emir Toktar, pelo apoio concedido à viabilização deste trabalho.

À PUC-PR pela oportunidade concedida.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

3DES	Triple DES (Data Encryption Standard)
AH	Authentication Header
CBC	Cipher Block Chaining
CIM	Common Information Model
CMIP	Common Management Information Protocol
DES	Data Encryption Standard
DMI	Desktop Management Interface
DMTF	Distributed Management Task Force
DNS	Domain Name System
DTD	Document Type Definition
ESP	Encapsulating Security Payload
HMAC	Hashed Message Authentication Code
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IP	Internet protocol
IPComp	Internet Protocol (IP) Compression
IPsec	Internet Protocol (IP) Security
ISAKMP	Internet Security Association and Key Management Protocol
KDC	Key Distribution Center
MD5	Message Digest 5
PCIM	Policy Core Information Model
PCIMe	Policy Core Information Model Extensions
PKI	Public-Key Infrastructure

PMI	Privilege Management Infrastructure
PRF	Pseudo-random function
RBAC	Role-Based Access Control
RC5	Rivest Cipher 5, também conhecido como Ron's Code 5
RSA	Rivest, Shamir, and Adleman – Nome dos inventores deste algoritmo de criptografia de chave pública
SA	Security Association
SAD	Security Associations Database
SHA-1	Secure Hash Algorithm 1
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SPD	Security Policy Database
SPI	Security Parameters Index
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifiers
UUID	Universally unique identifier
VPN	Virtual Private Network
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Resumo

Segundo [FUR02a] a segurança é o elemento mais importante em redes, principalmente em ambientes distribuídos através da Internet. Apesar de haver muito estudo sobre os *Web Services*, ainda não foi padronizado um mecanismo para descrição de segurança. Mesmo a linguagem WSDL, utilizada para descrever as interfaces dos *Web Services*, não trata de segurança.

O IETF desenvolveu um modelo genérico para representação de políticas (PCIM [MOO01] e PCIMe [MOO03]), e está atualmente criando uma especialização desse modelo para representar políticas IPsec [JAS02]. Entretanto, esses modelos são apenas estruturas de classes e relacionamentos entre elas, não havendo uma especificação de como as informações contidas nelas serão publicadas entre diferentes aplicações.

Assim, este trabalho define um esquema XML para representação de políticas IPsec para ser utilizado com os *Web Services*. Qualquer desenvolvedor que disponibilizar um *Web Service* poderá utilizar um documento WSDL para descrever as interfaces de seu serviço e também descrever as políticas de segurança IPsec para acessá-lo.

Além disso, também é proposta uma API para realizar a busca, interpretação e aplicação das políticas. Com o objetivo de prover portabilidade, a API foi desenvolvida utilizando a linguagem de programação Java e utiliza documentos de transformação XSL para tratamento dos documentos XML e criação dos comandos de configuração do dispositivo IPsec. Dessa forma, quando for necessário utilizar um dispositivo IPsec diferente, basta modificar o documento de transformação.

A API possui a funcionalidade de extrair somente as políticas que são aceitas pela política do cliente. Essa seleção de políticas é descrita através de um documento XSL chamado de Requisitos Mínimos, criado pelo cliente de acordo com suas políticas de segurança.

Palavras chave: 1. Segurança 2. Web Services 3. IPsec 4. WSDL

Abstract

According to [FUR02a], security is the most important element in business networks, particularly in distributed environments where data is exchanged over the internet. Although there is a lot of study about Web Services, today there isn't a standard way for describing security for it. Even the WSDL language, used for describing the interfaces of the services, doesn't deal with security.

IETF developed a generic model for policy representation (PCIM [MOO01] and PCIMe [MOO03]), and is currently working in a model to represent IPsec Policies [JAS02]. However, these models are only structures of classes and its relationship, there is no specification how to exchange these policies between different applications.

Thus, this work defines a XML schema for IPsec policies to be used with Web Services. By using the proposed approach, any developer that wants to deploy a Web Service will be able to use a WSDL document for describing the interfaces of the services and the IPsec policies necessary to access the services.

Moreover, an API is proposed to load, to interpret and to apply the policies. To make it portable, the API was developed in Java and uses XSL transformation documents to work with the XML documents and to create the configuration commands for the IPsec device. When is necessary to use a different IPsec device, the user only needs to modify the transformation document.

The API is able to extract only the policies accepted by the client policy. This selection is defined by a XSL document called Minimum Requirements, created by the client following its security policy.

Keywords: *1. Security 2. Web Services 3. IPsec 4. WSDL.*

Capítulo 1

Introdução

1.1. Desafio

O uso da Internet por organizações para realização de negócios está em crescimento constante. As organizações têm buscado cada vez mais conectar seus sistemas diretamente aos sistemas de seus fornecedores e consumidores, sem a necessidade de interferência humana, objetivando diminuição de custos e aumento na velocidade [MIY00].

Uma arquitetura que está sendo utilizada nessa integração é a de *Web Services* [ROY01], que consiste de serviços que podem ser acessados através de padrões como HTTP [BER96] e XML [BRA00] [BER00]. Os serviços podem ser disponibilizados a qualquer pessoa ou organização interessada, ou a um grupo escolhido a partir de certos critérios, por exemplo os atuais parceiros de negócio.

Quando as mensagens são transportadas através da Internet, faz-se necessário implementar mecanismos de segurança, pois os dados podem ser roubados, perdidos ou modificados. Estas mensagens podem representar transações entre organizações e falhas na segurança podem representar perdas de grandes valores e problemas para a reputação das organizações. Se não houvesse segurança, a Internet não teria utilidade para realização de transações [XAV01].

Apesar de existirem muitos mecanismos de segurança, não há uma forma padronizada para negociar as restrições de segurança que os clientes e servidores devem atender. [MAL03] acredita que mesmo até o final de 2003 isso ainda não estará totalmente padronizado.

1.2. Motivação

Segundo [FER02], 51% dos participantes da conferência *InfoWorld Next Gen Web Services* consideram que o maior obstáculo para a aceitação geral dos *Web Services* é a segurança.

A segurança engloba os seguintes requisitos [NAK02] [DOW98]:

- Confidencialidade: somente o remetente e o destinatário têm acesso ao conteúdo da mensagem.
- Autenticação: garante que o acesso às aplicações e aos dados é restrito somente àqueles que podem fornecer uma prova de identidade apropriada.
- Integridade: protege a mensagem contra modificações acidentais ou propositais.
- Não-repudiação: o remetente não pode negar que enviou a mensagem.

Alguns mecanismos atendem somente alguns requisitos, mas podem ser combinados com outros mecanismos para obter a segurança desejada.

Atualmente existem vários mecanismos para prover segurança. Eles se resumem a duas abordagens: estabelecer comunicações seguras implementadas na pilha de protocolos de rede e a proteção de cada mensagem ou mesmo partes da mensagem, através da criação de um envelope dentro do qual a mensagem é protegida.

Mas apesar dos vários mecanismos existentes, nenhum deles define uma forma de representar as políticas de segurança, de forma que suas políticas possam ser fornecidas aos clientes e que esses possam de forma automática realizar as configurações necessárias em seus sistemas e trocarem mensagens sem intervenção humana.

Com o objetivo de representar políticas genéricas, o IETF desenvolveu o *Policy Core Information Model* (PCIM) [MOO01], que é um modelo orientado a objetos para representação de políticas de qualquer tipo. Esse é um modelo genérico; para representar um determinado mecanismo de segurança é necessário criar extensões que representem as informações deste mecanismo.

Derivado desse modelo, está sendo desenvolvido um modelo específico para representação de políticas IPsec [JAS02]. Esse modelo possui as classes necessárias para representar todos os parâmetros para a criação de canais IPsec. Entretanto, esse é apenas um modelo de classes, não possuindo nenhum mapeamento para uma representação que possa ser trocada entre clientes e servidores.

1.3. Proposta

O objetivo deste trabalho é desenvolver um mecanismo para permitir transações seguras entre aplicações que utilizam *Web Services*. Esse mecanismo é composto por um esquema XML para representação de políticas de segurança baseado em IPsec, uma API[1] para interpretar esses documentos XML e aplicar as políticas no sistema e um documento de transformação XSL[2] de requisitos mínimos de segurança, definido de acordo com as políticas do cliente.

De acordo com a proposta, um servidor de *Web Services* disponibiliza, juntamente com a descrição dos serviços oferecidos, um conjunto de políticas IPsec que devem ser utilizadas para acessar esses serviços de forma segura. O cliente utiliza essas informações para criar em seu equipamento as políticas IPsec necessárias para acessar o serviço. Como as políticas de segurança do cliente podem ser mais rígidas que as fornecidas pelo servidor, o cliente, utilizando seu documento XSL de requisitos mínimos, tem a opção de selecionar entre as políticas fornecidas quais são aceitas. As políticas selecionadas são convertidas para comandos do dispositivo IPsec, que serão executados no momento em que for necessário realizar a chamada ao servidor de *Web Service*.

Com o objetivo de se adequar aos padrões existentes, esse trabalho também propõe a extensão dos documentos de descrição dos serviços (WSDL), de forma a representar as políticas como parte integrante dessa descrição de serviços.

Por fim, essa proposta define uma API para manipular o arquivo de descrição de políticas. Essa API será utilizada por aplicações clientes e fará a conversão da política de segurança para comandos do sistema operacional.

1.4. Justificativa

O IPsec é um mecanismo destinado a prover proteção na transmissão de mensagens. Apesar de dispor de recursos para realizar a negociação dos parâmetros de segurança, é necessário que o dispositivo já possua toda a configuração para que a negociação possa ser iniciada. O problema é que não há uma forma padronizada de fornecer informações de configuração dos dispositivos. O modelo de políticas IPsec do IETF define uma estrutura

---

[1] API – Application Program Interface: Interface para programação de aplicações.
[2] XSL – Extensible Stylesheet Language [ADL01]

para armazenamento das políticas, mas não descreve como essas informações podem ser fornecidas.

Por esse motivo, é necessário um esquema que permita a representação das políticas de forma que elas possam ser publicadas e que qualquer um seja capaz de interpretá-las e utilizá-las para configurar seus dispositivos IPsec. O XML foi escolhido para representar as políticas pois todos os protocolos utilizados pelos *Web Services* são baseados nele. Dessa forma, mantém-se uma padronização, além de tornar possível a extensão do documento de descrição de interfaces (WSDL) para conter informações sobre as políticas de segurança.

1.5. Organização

Esta dissertação está estruturada em oito capítulos, organizados da seguinte forma:

Capítulo 1: Introdução: Apresenta o contexto geral deste trabalho.

Capítulo 2: Padrões Utilizados em *Web Services*: Descreve os principais conceitos e protocolos utilizados para a implementação de *Web Services*.

Capítulo 3: Análise de Mecanismos de Segurança para *Web Services*: Apresenta mecanismos de segurança mais indicados para a utilização com os *Web Services*. Descreve com detalhes o IPsec, protocolo este que foi utilizado neste trabalho.

Capítulo 4: Modelos IETF para Representação de Políticas IPsec: Esses modelos são estruturas de classes para representação de políticas IPsec. Serão utilizados como base para o desenvolvimento da extensão do WSDL para políticas de segurança.

Capítulo 5: Análise do Uso do IPsec com *Web Services*: Apresenta a análise de vários aspectos que devem ser considerados na utilização do IPsec com *Web Services*.

Capítulo 6: Proposta de Extensão do WSDL para Suportar IPsec: Apresenta a extensão do WSDL, bem como a API desenvolvida para trabalhar com ela.

Capítulo 7: Estudo de Caso: Apresenta uma implementação de um cliente e um servidor de *Web Service*, e testes para avaliar o impacto causado pelo uso do esquema, da API e o IPsec no desempenho de chamadas ao *Web Service*.

Capítulo 8: Conclusões e Trabalhos Futuros: Apresenta as conclusões obtidas nesse trabalho e possíveis trabalhos para a continuação do mesmo.

Capítulo 2

Padrões Utilizados em Web Services

2.1. Introdução ao Capítulo

Quando a Internet começou a se popularizar, as tecnologias da época permitiam aos usuários conectarem-se a um *site* na Internet e obter o conteúdo deste. A linguagem HTML e o protocolo HTTP eram suficientes para esse fim.

À medida que a Internet foi crescendo, observou-se que ela poderia ser utilizada para aplicações mais complexas. Uma dessas aplicações é a disponibilização de serviços eletrônicos, através dos quais é possível realizar o comércio eletrônico.

Os *Web Services* provêem uma forma padronizada de interoperabilidade entre diferentes aplicações, que podem ser implementadas em diferentes linguagens e estar executando em diferentes plataformas. Um *Web Service* é identificada por um endereço (tipicamente uma URI - *Uniform Resource Identifier* [BER98]), com interfaces públicas e definidas através da linguagem XML

Um motivo para que os *Web Services* sejam uma grande aposta do mercado é a possibilidade de integração dos sistemas legados. Especialmente para as grandes organizações, que possuem diversas soluções de fornecedores diferentes ou mesmo desenvolvidos internamente nas mais variadas plataformas. Com os *Web Services*, a promessa é que todas as informações contidas nas aplicações possam ser unificadas pela Web, sem que haja a necessidade de migração.

Este capítulo apresenta as principais tecnologias utilizadas para o desenvolvimento e utilização dos *Web Services*. A seção 2.2 apresenta a arquitetura dos *Web Services*. A seção 2.3 apresenta o protocolo SOAP, utilizado em *Web Services* para evocar

procedimentos remotos. A seção 2.4 apresenta o WS-Routing, utilizado para indicar o caminho que a mensagem SOAP deve seguir para chegar ao destinatário. A seção 2.5 apresenta a WSDL, utilizada para descrever as interfaces disponibilizadas pelos *Web Services*. A seção 2.6 apresenta o UDDI, que é uma proposta de para um mecanismo de publicação e localização de serviços de forma dinâmica. A seção 2.7 conclui o capítulo.

2.2. Web Services

Um *Web Service* é um componente ou unidade de software que fornece uma funcionalidade específica, que pode ser acessada por diferentes sistemas através do uso de linguagens e protocolos padronizados, como XML (*Extensible Markup Language*) [BRA00] [BER00] e HTTP (*Hyper Text Transfer Protocol*) [BER96].

A utilização de tais padrões é fundamental para o desenvolvimento de aplicações distribuídas com *Web Services*. Um *Web Service* poderá ser utilizado internamente por uma única aplicação ou por várias aplicações da mesma organização. O mesmo *Web Service* pode ser exposto através da Internet, a fim de ser utilizado por qualquer aplicação, bastando para isso que a aplicação seja capaz de entender SOAP (*Simple Object Access Protocol*) [BOX00] e XML.

Para permitir que as aplicações clientes possam descobrir de forma dinâmica a interface do *Web Service*, os serviços devem ser descritos utilizando-se a linguagem WSDL [CHR01] (*Web Services Description Language*). Essa linguagem permite publicar as informações sobre a sintaxe, métodos, parâmetros e localização dos serviços. Dessa forma, para construir uma aplicação que realize chamadas para um determinado *Web Service*, obtém-se o arquivo WSDL associado e com base nas informações contidas nele desenvolve-se a aplicação.

Os documentos WSDL podem ser armazenados no UDDI [BEL02] (*Universal Description, Discovery and Integration*), que é uma proposta para diretório geral para registro e pesquisa de serviços disponíveis. Os clientes interessados em um determinado serviço podem fazer uma busca nesse diretório e obter as informações relativas à interface e à localização do serviço.

A maior vantagem do *Web Services* é que ele é um padrão completamente independente da tecnologia usada para construir aplicativos. Os *Web Services* estão acima de plataformas, bancos de dados e linguagens de programação, eliminando as limitações

existentes em outras interfaces entre aplicativos. Portanto, os clientes podem ser implementados em qualquer plataforma e em qualquer linguagem de programação, independente da plataforma e da linguagem nas quais foram implementados os servidores. [CHE01] apresenta exemplos de *Web Service* implementados em ASP e em PERL, e utilizando as plataformas Windows 2000, IBM OS/390 e SUN Solaris.

A arquitetura típica dos *Web Services* consiste de 3 entidades (Figura 2.1):

- *Service providers*: criam os *Web Services* e os publicam ao mundo externo registrando os serviços em *Service Brokers*;
- *Service brokers*: mantém o registro de serviços publicados;
- *Service Requesters*: procuram por serviços consultando o registro dos *service brokers*. Após encontrar o serviço desejado eles fazem a conexão com os *service providers* para utilizar o serviço.

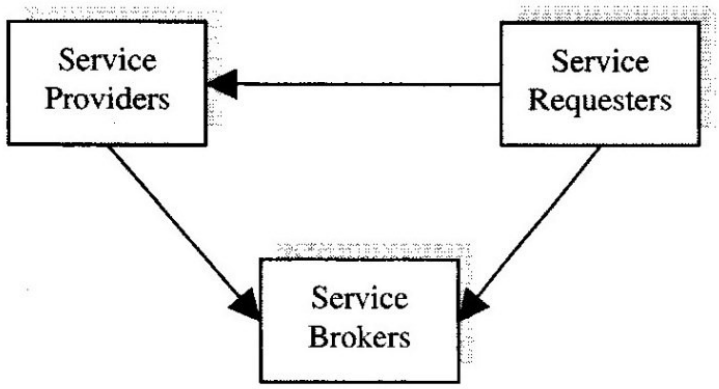

Figura 2.1: Arquitetura dos Serviços Web

Para que o *Service Requester* possa se comunicar com o *Service Provider*, ele precisa conhecer as interfaces e a localização do *Service Provider*. Esse conhecimento pode ser obtido através de documentos físicos (como um contrato de serviço) ou através de um documento WSDL. O documento WSDL pode ser obtido diretamente do provedor do serviço ou então através do *Service Broker* (nesse caso pode-se utilizar o protocolo UDDI). A partir do WSDL o cliente deve criar as chamadas e enviá-las ao *Service Provider*. Toda a interação entre as entidades é realizada utilizando-se o protocolo SOAP. Esses protocolos são apresentados com mais detalhes nas seções seguintes.

2.3. SOAP – Simple Object Access Protocol

O SOAP é um protocolo simples para troca de mensagens XML pela Web. A especificação define um envelope para transmissão de mensagens, oferece diretrizes para codificação dos dados e provê regras para representação de chamada de procedimentos remotos (RPCs[3]). O SOAP está se transformando no padrão para troca de mensagens XML [ROY01].

A especificação permite o transporte de mensagens XML através de protocolos de alto nível como HTTP, SMTP e outros. O protocolo HTTP é o mais utilizado pois pode atravessar facilmente os *firewalls* [SKO00]. Por utilizar protocolos padronizados, o SOAP pode ser utilizado para interoperabilidade entre diferentes plataformas [JEP01].

As mensagens SOAP são mensagens XML definidas dentro de um envelope SOAP. O envelope contém um corpo obrigatório e um cabeçalho opcional. O corpo contém a mensagem XML a ser transmitida. O cabeçalho, se presente, tipicamente contém informações relacionadas à segurança, roteamento ou informações ao destinatário de tratamento da mensagem.

O SOAP consiste de quatro partes:

- O envelope SOAP define a base para expressar o que é a mensagem, quem deve manipulá-la e se a manipulação de cada elemento é opcional ou obrigatória;
- As regras de codificação SOAP, que definem o mecanismo de serialização a ser utilizado para trocar tipos de dados entre aplicações;
- A representação RPC que define a convenção a ser utilizada nas chamadas de procedimentos remotos e suas respostas.
- A ligação SOAP, que define uma convenção para a troca de envelopes SOAP entre computadores utilizando um protocolo para transporte.

As mensagens SOAP são fundamentalmente transmissões de um sentido só, do remetente ao destinatário. Duas ou mais mensagens podem ser combinadas para implementar padrões como solicitação/resposta.

É chamado de nó SOAP o remetente inicial, o último destinatário ou um intermediário. O SOAP permite indicar quais partes das mensagens são destinadas para um

---

[3] Remote Procedure Call (RPC): chamada de procedimento remoto. Permite que um programa execute procedimentos e/ou funções que estão disponibilizados em outro computador.

determinado nó intermediário. Quando um nó SOAP estiver processando uma mensagem, diz-se que esse nó está agindo na função de um ou mais atores SOAP, cada um deles identificado por uma URI chamada de nome do ator SOAP. O propósito do nome do ator SOAP é identificar um nó SOAP. Apesar de permitir a existência de nós intermediários, o SOAP não define um mecanismo de roteamento.

A Figura 2.2 apresenta um exemplo de uma chamada utilizando SOAP, transmitida através do protocolo HTTP. Essa mensagem evoca o método GetLastTradePrice. Este método requer como parâmetro o nome da companhia e um símbolo que representa o produto, e retorna um valor número representando o último preço de um determinado produto.

```
POST StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
     <m:GetLastTradePrice xmlns:m="Some-URI">
       <company>XYZCompany</company>
       <symbol>MOT</symbol>
     </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 2.2: Exemplo de uma Requisição SOAP utilizando HTTP

O envelope SOAP, representado pelo elemento "<SOAP-ENV:Envelope ...>", é o elemento principal da mensagem. Esse envelope contém somente o corpo da mensagem, representado por "<SOAP-ENV:Body>", que contém o elemento representando a chamada do método GetLastTradePrice. Os dois parâmetros requeridos são passados nessa chamada, o nome companhia é XYZCompany e o símbolo do produto é MOT.

A Figura 2.3 é o exemplo da resposta à solicitação feita ao *Web Service* na Figura 2.2. De forma semelhante à solicitação, a resposta contém um envelope composto somente pelo corpo da mensagem. No corpo da mensagem aparece a resposta do chamado, representado pelo elemento GetLastTradePriceResponse. O valor do produto retornado como resposta é 14.5.

```
HTTP 1.1 200 OK Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
 <SOAP-ENV:Body>
   <m:GetLastTradePriceResponse xmlns:m="Some-URI">
     <Price>14.5</Price>
   </m:GetLastTradePriceResponse>
 </ SOAP-ENV:Body>
</ SOAP-ENV:Envelope>
```

Figura 2.3: Exemplo de uma Resposta SOAP utilizando HTTP

2.4. WS-Routing

O protocolo SOAP introduz um modelo de processamento distribuído no qual algumas partes da mensagem podem ser destinadas aos nós intermediários. Entretanto, a especificação do SOAP não define uma semântica de roteamento ou troca de mensagens, que descreva o caminho da mensagem SOAP. Por exemplo, supondo quatro nós SOAP, A, B, C e D, uma mensagem gerada por A com destino a D pode indicar quais partes da mensagem são destinadas para cada intermediário, mas não pode indicar que a mensagem deve ser enviada para D através dos intermediários B e C.

O protocolo de roteamento SOAP (WS-Routing) [NIE01] é um protocolo baseado no SOAP para troca de mensagens desde o remetente até o último destinatário, potencialmente através de um conjunto de nós intermediários. Além disso, o WS-Routing pode opcionalmente indicar o caminho inverso permitindo a troca de mensagens nos dois sentidos (solicitação/resposta) e também o retorno de mensagens de recebimento ou falhas.

A especificação do WS-Routing define os seguintes mecanismos e conceitos:

- O caminho de transferência de mensagens adiante;
- O caminho reverso de transferência de mensagens (opcional);
- Um mecanismo de correlação entre mensagens.

A especificação define uma entrada para o cabeçalho SOAP, que descreve o caminho de transferência (e o reverso, se for o caso) para a mensagem SOAP. O nó remetente indica qual é o destinatário final da mensagem e zero ou mais intermediários (que também são chamados de destinatários). Essa mensagem é transferida para o primeiro destinatário através de algum protocolo de transporte (HTTP, SMTP ou outro). Se o nó que estiver recebendo a mensagem for um nó intermediário, ele reenvia a mensagem para o próximo destinatário até que a mensagem chegue ao destinatário final ou uma falha ocorra.

Os nós intermediários podem dinamicamente inserir intermediários adicionais. Isto é, não há nenhum requerimento que todo o caminho da mensagem seja conhecido no momento que o remetente envia a mensagem.

Um elemento "rev" é utilizado para representar o caminho reverso. Somente o remetente pode incluir o elemento "rev". O caminho reverso é construído dinamicamente à medida que a mensagem atravessa os nós intermediários (cada nó acrescenta uma entrada de caminho reverso). Caso o nó intermediário não possa determinar o caminho reverso, este deve gerar uma falha do tipo caminho reverso não disponível.

A Figura 2.4 é o exemplo de uma mensagem SOAP contendo informações sobre o caminho pelo qual a mensagem deve passar. A mensagem foi criada pelo nó A, com destinatário final D (representado no elemento <to>), e caminho passando por B e C (representados pelo elemento <via>).

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://www.im.org/chat</m:action>
         <m:to>soap://D.com/some/endpoint</m:to>
         <m:fwd>
            <m:via>soap://B.com</m:via>
            <m:via>soap://C.com</m:via>
         </m:fwd>
         <m:rev>
            <m:via/>
         </m:rev>
         <m:from>mailto:henrikn@microsoft.com</m:from>
         <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
      </m:path>
   </S:Header>
   <S:Body>
      ...
   </S:Body>
</S:Envelope>
```

Figura 2.4: Exemplo de uma mensagem com cabeçalho WS-Routing

A Figura 2.5 apresenta a mesma mensagem após passar pelo nó B. Nesse exemplo pode-se observar que no elemento <fwd> foi removida a informação de que o caminho passa pelo nó B e no elemento <rev> foi incluído o caminho reverso.

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://www.im.org/chat</m:action>
         <m:to>soap://D.com/some/endpoint</m:to>
         <m:fwd>
            <m:via>soap://C.com</m:via>
         </m:fwd>
         <m:rev>
            <m:via/>
            <m:via m:vid="cid:122326@B.com"/>
         </m:rev>
         <m:from>mailto:henrikn@microsoft.com</m:from>
         <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
      </m:path>
   </S:Header>
   <S:Body>
      ...
   </S:Body>
</S:Envelope>
```

Figura 2.5: Mensagem com cabeçalho WS-Routing após passar pelo primeiro intermediário

2.5. WSDL - Web Service Description Language

Trata-se de uma linguagem de definição de interface, baseada em XML, a qual possibilita a descrição das funcionalidades disponibilizadas pelo *Web Service* utilizando um formato padrão, de forma que outra aplicação possa compreendê-la. A WSDL é considerada como um dos fundamentos que possibilita a especificação consistente de um *Web Service*.

Através da WSDL, são descritos os métodos e atributos suportados pelo *Web Service*, os tipos de dados e os protocolos que podem ser utilizados para o envio e recebimento de mensagens. As informações inseridas nas mensagens são repassadas por meio de operações e palavras-chaves da linguagem WSDL, como type, message, operation ou binding. A linguagem define somente as interfaces necessárias para a chamada das funções. A lógica de negócio e a linguagem de implementação não fazem parte da definição.

Conceitualmente, essa linguagem é similar ao protocolo RMI (utilizado pelo J2EE) e à linguagem IDL (*Interface Definition Language*, utilizado pelo CORBA).

A WSDL define serviços como coleções de *endpoints* de rede ou portas[4] (*ports*). Na WSDL, a definição abstrata de *endpoints* e mensagens é separada da configuração de rede e dos formatos de dados. Isso permite a reutilização das definições de mensagens (descrições abstratas de dados a serem trocados) e tipos de portas (coleções abstratas de operações). O

[4] Em Web Services, a representação de portas é diferente da utilizada para conexões IP. Nos dois casos uma porta representa um serviço em uma máquina, mas ao invés de um número como utilizado em conexões IP, em Web Services a porta é a combinação de um *binding* com um endereço de rede (URI), representando um determinado serviço em uma máquina.

protocolo concreto e a especificação de formatos de dados para um tipo de porta em particular constituem um *binding*[5] reutilizável. Uma porta é definida associando um endereço de rede a um *binding* reutilizável, e uma coleção de portas define um serviço. Assim, um documento WSDL utiliza os seguintes elementos na definição de serviços de rede:

- Types: um contêiner para definições de tipos de dados utilizando algum sistema de tipos (como *XML Schema Definition* – XSD [FAL01] [THO01] [BIR01]);
- Message: uma definição tipada abstrata de dados sendo comunicados. Uma mensagem consiste de uma ou mais partes, sendo que cada parte representa um parâmetro tipado;
- Operation: uma descrição abstrata de ações suportadas pelo serviço, normalmente composta por uma mensagem de entrada (com os parâmetros) e uma mensagem de saída (com a resposta);
- Port Type: um conjunto abstrato de operações e mensagens suportadas por um ou mais *endpoint*;
- Binding: especifica um protocolo concreto e formatos de dados para um Port Type particular. Descreve os detalhes técnicos de implementação do *Web Service*.
- Port: um *endpoint* definido como a combinação de um *binding* com um endereço de rede;
- Service: uma coleção de *endpoint* relacionados. Define a localização do *Web Service*, através de uma URI (*Uniform Resource Identifiers*) [BER98].

O arquivo apresentado na Figura 2.6 é a estrutura genérica de um arquivo WSDL.

[5] Binding: especifica a ligação de um conjunto de operações e mensagens com um protocolo concreto e formato de dados.

```
<?xml version="1.0"?>
<definitions name="StockQuote"
             targetNamespace="http://example.com/stockquote.wsdl"
             xmlns:tns="http://example.com/stockquote.wsdl"
             xmlns:xsd1="http://example.com/stockquote.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
  < types>

  <message name="...">
  < message>

  <portType name="...">
    <operation name="...">
    </operation>
  < portType>

  <binding name="StockQuoteSoapBinding"
           type="tns:StockQuotePortType">
  < binding>

  <service name="StockQuoteService">
  < service>

< definitions>
```

Figura 2.6: Estrutura genérica de uma definição WSDL

Para criar um arquivo WSDL completo, deve-se preencher essa estrutura de acordo com o(s) serviço(s) disponíveis. Uma possível representação de um serviço Web que forneça o preço de um determinado produto é a seguinte:

- São declarados dois tipos de mensagens, um é TradePriceRequest e o outro é TradePrice (Figura 2.7). A primeira possui um parâmetro chamado tickerSymbol, que é uma identificação do produto do qual se quer saber o preço. A segunda possui um parâmetro price que indica o preço do produto.

```
<types>
  <schema targetNamespace="http://example.com/stockquote.xsd"
          xmlns="http://www.w3.org/1999/XMLSchema">
   <element name="TradePriceRequest">
    <complexType>
     <all>
      <element name="tickerSymbol" type="string"/>
     </all>
    </complexType>
   </element>
   <element name="TradePrice">
    <complexType>
     <all>
      <element name="price" type="float"/>
     </all>
    </complexType>
   </element>
  </schema>
</types>
```

Figura 2.7: Declaração de tipos de mensagens

- A partir dos dois tipos de mensagens, são definidas duas mensagens, uma de cada tipo, com os nomes GetLastTradePriceInput e GetLastTradeOutput (Figura 2.8).

```
<message name="GetLastTradePriceInput">
  <part name="body" element="xsd1:TradePriceRequest"/>
</message>
<message name="GetLastTradePriceOutput">
  <part name="body" element="xsd1:TradePrice"/>
</message>
```

Figura 2.8: Definição de mensagens

Na seqüência define-se um portType com o nome StockQuotePortType, que representa o conjunto de operações disponibilizadas pelo serviço Web (Figura 2.9). É definida uma operação, GetLastTradePrice, sendo que esta operação possui as duas mensagens definidas anteriormente, GetLastTradePriceInput e GetLastTradeOutput.

```
<portType name="StockQuotePortType">
 <operation name="GetLastTradePrice">
  <input message="tns:GetLastTradePriceInput"/>
  <output message="tns:GetLastTradePriceOutput"/>
 </operation>
</portType>
```

Figura 2.9: Definição do tipo de porta

- O próximo passo é definir a ligação do portType com um protocolo, no caso HTTP (Figura 2.10). A ligação define o esquema, o tipo de

codificação e outros detalhes que variam de acordo com o protocolo utilizado. Nesse arquivo a ligação recebeu o nome de StockQuoteSoapBinding.

```
<binding name="StockQuoteSoapBinding"
         type="tns:StockQuotePortType">
<soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
   <soap:operation
    soapAction="http://example.com/GetLastTradePrice"/>
     <input>
       <soap:body use="literal"
        namespace="http://example.com/stockquote.xsd"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </input>
    <output>
     <soap:body use="literal"
       namespace="http://example.com/stockquote.xsd"
       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
   </operation>
</binding>
```

Figura 2.10: Definição do elemento *Binding*

- Por fim, o serviço é definido com o nome StockQuoteService, e contém uma porta chamada StockQuotePort que utiliza a ligação definida no passo anterior, e indica o endereço a ser utilizado para acessar este serviço (Figura 2.11).

```
<service name="StockQuoteService">
 <documentation>My first service</documentation>
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">
   <soap:address location="http://example.com/stockquote"/>
  </port>
</service>
```

Figura 2.11: Definição de serviço

A WSDL permite que novos elementos sejam inseridos na linguagem com o objetivo de estendê-la. A especificação da WSDL [CHR01] define algumas possibilidades de extensão, que foram utilizadas nesse trabalho com o objetivo de acrescentar a descrição das políticas de segurança.

2.6. UDDI – Universal Description, Discovery and Integration

O UDDI foi desenvolvido para permitir que as organizações realizem transações entre si de maneira rápida, fácil e dinâmica. Ou seja, que possam encontrar os serviços desejados e possam se comunicar. Como resultado, foi desenvolvida uma especificação para um registro central de acesso e controle.

Essa especificação descreve como os dados devem ser armazenados em um registro, além de definir os métodos possíveis para publicação e busca desses registros. Os dados incluem os contatos da organização, como se comunicar com eles, uma lista de serviços disponíveis e como usá-los por meio de algum tipo de programação. Todos os acessos ao registro são realizados utilizado o padrão SOAP tanto para consulta como para atualização.

Uma analogia para o UDDI é "lista telefônica para *Web Services*" [UDD01].

Conceitualmente, as informações disponíveis no registro do UDDI consistem de três componentes:

- "Páginas brancas": inclui os detalhes como nome e informações para contato;
- "Páginas amarelas": provê a categorização baseada nos tipos de serviço e negócio, baseado em taxonomias;
- "Páginas verdes": inclui dados técnicos sobre os serviços.

O registro UDDI é organizado em duas entidades fundamentais que descrevem o negócio e os serviços providos. O elemento businessEntity (Figura 2.12) provê informações padrões para as páginas brancas. Cada elemento businessEntity pode incluir um ou mais elementos businessServices, que representa os serviços providos, apresentado na Figura 2.13. As duas entidades (negócio e serviço) podem conter um elemento categoryBag para indicar a categoria do negócio ou serviço.

```
<businessEntity businessKey="A687FG00-12EM-5432-012345678932">
   <name>Acme Travel Incorporated</name>
   <description xml:lang="en"> Online travel services </description>
   <contacts>
      <contact useType="US General">
         <personName>Acme Inc.</personName>
         <phone>1 800 CALL ACME</phone>
         ...
      </contact>
   </contacts>

   <businessServices> ...
   </businessServices>

   <identifierBag> ...
   </identifierBag>

   <categoryBag> ...
      <keyedReference tModelKey="UUID:DB123456-9ABC-DD12-41411DF3E235"
         keyName="Eletronic check-in"
         keyValue="84121801"/>
   </categoryBag>
</businessEntity>
```

Figura 2.12: Estrutura businessEntity simplificada

A entidade de serviço contém uma lista de bindingTemplates que codificam as informações técnicas de acesso aos serviços, sendo que cada elemento bindingTemplate representa um ponto de acesso para o serviço.

Um detalhe importante a se observar é a utilização de identificadores únicos universais (UUID - *Universally Unique Identifier*). Esses identificadores (também chamados de chaves) são longas cadeias de caracteres hexadecimais gerados quando a entidade é registrada e devem ser únicos (não podem ser utilizados para representar outras entidades nem mesmo em outros registros UDDI). Por exemplo, o atributo `businessKey` (Figura 2.12) identifica unicamente uma entidade de negócio e o atributo serviceKey (Figura 2.13) identifica um serviço.

```
<businessService serviceKey="893A2100-12EM-5432-012345678932"
                 businessKey="A687FG00-12EM-5432-012345678932">
   (...)
   <bindingTemplates>

      <bindingTemplate>
         (...)
         <accessPoint urlType="http">http://www.etc.com/</acessPoint>
         <tModelInstanceDetails>
            <tModelInstaceInfo tModelKey="...">

            </tModelInstanceInfo>
         </tModelIntanceDetails>
      </bindingTemplate>
      (...)

   </bindingTemplates>
</businessService>
```

Figura 2.13: Estrutura businessService Simplificada

O elemento mais interessante é o tModelInstanceDetails, que contém uma lista de referências para as descrições técnicas dos serviços (as páginas verdes). O provedor do serviço registra as especificações técnicas requeridas no diretório, sendo que cada uma recebe um identificador único. Cada especificação é registrada por uma entidade de informação chamada tModel (Figura 2.14). Qualquer serviço que suporte a mesma especificação pode simplesmente adicionar uma referência ao elemento `tModel` em sua lista tModelInstanceDetails.

```
<tModel tModelKey="">
   <name>http://www.travel.org/e-checkin-interface</name>
   <description xml:lang="en">Travel information</description>
   <overviewDoc>
      <description xml:lang="en">
         WSDL Service Interface Document
      </description>
      <overviewURL>
         http://www.travel.org/services/e-checkin.wsdl
      </overviewURL>
   </overviewDoc>
   <categoryBag>
      <keyedReference tModelKey="UUID:C1ACF26D-9672-4404-39B756E62AB4"
                      keyName="uddi-org:types"
                      keyValue="wsdlSpec"/>
   </categoryBag>
</tModel>
```

Figura 2.14: Exemplo da Estrutura tModel

Por exemplo, suponha um documento WSDL registrado como um tModel. Assumindo que o segmento de companhias de viagem defina interfaces padrões para obtenção de informações de vôo e *check-in* eletrônico, primeiro é necessário criar um tModel como o

apresentado na Figura 2.14 para representar essas definições. Os serviços que implementam essas interfaces podem incluir o tModel correspondente em sua lista tModelInstanceDetails. Obviamente, para isso funcionar, usuários e desenvolvedores devem concordar com as interfaces e suas chaves.

Para identificar e permitir a localização de tipos particulares de negócios e serviços, é necessário qualificar os serviços de acordo com um esquema de categorização ou uma taxonomia. Exemplos de categorias são a localização geográfica e o tipo de indústria ou produto. Cada sistema de taxonomia é classificado como um tModel no registro do UDDI. As informações de taxonomia são codificadas como pares nome-valor, qualificados por uma referência tModel que identifica a qual taxonomia cada par pertence.

2.6.1. Modelo de Invocação UDDI

Cada *Web Service* publicado é modelado com a estrutura bindingTemplate. Cada invocação do serviço é realizada baseada nos dados dessa estrutura, que foi obtida anteriormente e está armazenada. Com isso em mente, o cenário geral para utilização do UDDI é o seguinte:

1. O programador usa o registro do UDDI (através de uma interface WEB ou alguma ferramenta que utiliza a API do UDDI) para localizar as informações do businessEntity.
2. O programador busca o businessService desejado ou obtém toda a estrutura businessEntity e armazena localmente.
3. O programador prepara o programa baseado na especificação do *Web Service*. Essa informação é obtida utilizando as informações do tModel contido no bindingTemplate.
4. No momento da chamada do *Web Service*, o sistema faz a chamada do serviço utilizando as informações do bindingTemplate armazenado.

No caso geral, assumindo que tanto o *Web Service* quanto a aplicação que o está utilizando foram implementados corretamente e seguem as convenções especificadas no tModel, a chamada ocorrerá com sucesso. Para os casos em que ocorrem falhas na chamada, o UDDI provê uma convenção que envolve utilizar as informações armazenadas anteriormente, e quando uma falha ocorre, atualizar as informações com o registro UDDI. O cenário da realização de uma chamada envolvendo tratamento de falhas é o seguinte:

1. Preparar o programa para chamadas *Web Service*, armazenando localmente os dados do bindingTemplate para utilização no momento da execução;
2. Quando uma chamada for realizada, utilizar as informações locais do bindingTemplate;
3. Se a chamada falhar, utilizar o valor bindingKey e a chamada da API get_bindingTemplate para obter uma cópia atualizada do bindingTemplate único para esse serviço;
4. Comparar a informação nova com a antiga, se for diferente, tentar novamente fazer a chamada. Caso a nova tentativa tenha sucesso, substituir a informação antiga pela nova informação.

2.7. Conclusão do Capítulo

Nesse capítulo foram apresentados os protocolos utilizados para representação, troca de mensagens e roteamento de mensagens utilizadas pelos *Web Services*.

Esses protocolos não prevêem métodos que garantam a privacidade das informações, e nem a autenticação para o acesso aos *Web Services*. Qualquer pessoa mal intencionada pode interceptar mensagens e descobrir o seu conteúdo, sendo possível ainda modificar o conteúdo e reenviá-lo, sem que o destinatário tenha conhecimento disso. Por isso, os *Web Services* precisam ser combinados com algum mecanismo de segurança.

O capítulo seguinte apresenta métodos que podem ser utilizados para prover a segurança, que incluem opções de criptografia, autenticação, autorização ou uma combinação destas opções. Alguns dos métodos podem ser utilizados para qualquer tipo de transmissão de dados, enquanto outros foram desenvolvidos especificamente para utilização com *Web Services*.

Capítulo 3

Análise de Mecanismos de Segurança para Web Services

3.1. Introdução ao Capítulo

Diversos mecanismos podem ser utilizados para a proteção de informações que são transmitidas pela Internet. Apesar disso, estão sendo estudados e desenvolvidos novos mecanismos a cada dia, buscando melhorar e otimizar suas características e funcionalidades.

Com o advento do XML, buscou-se criar mecanismos que permitam proteger mensagens individuais, ou até mesmo partes de uma mensagem. Dessa forma, é possível proteger apenas os campos que são importantes.

Dependendo das características do mecanismo de proteção, um ou mais dos requerimentos podem ser atendidos [NAK02] [DOW98]:

- Confidencialidade: somente o remetente e o destinatário têm acesso ao conteúdo da mensagem.
- Autenticação: garante que quem está enviando a mensagem é que diz ser.
- Integridade: protege a mensagem contra modificações acidentais ou propositais.
- Não-repudiação: o remetente não pode negar que enviou a mensagem.

Esse capítulo apresenta os mecanismos de proteção de mensagens mais comuns para utilização com *Web Services*, com o objetivo de mostrar vantagens e desvantagens de cada método e justificar a escolha do método utilizado neste trabalho. As seções 3.2 e 3.3 apresentam métodos que realizam a proteção de mensagens individuais, ou mesmo partes de uma mensagem. As seções 3.4 e 3.5 apresentam métodos que estabelecem comunicações

seguras implementadas na pilha de protocolos de rede. As seções 3.6 e 3.7 apresentam trabalhos que utilizam métodos já existentes para criar uma infra-estrutura que realizará a proteção das mensagens. Por fim, na seção 3.8 são apresentados de forma resumida outros trabalhos na área.

3.2. XML Encryption

O objetivo do XML Encryption [IMA02] é desenvolver um processo para cifrar e decifrar conteúdo digital (incluindo documentos XML e porções destes) e desenvolver uma sintaxe XML para representar o conteúdo cifrado e a informação que permitirá ao destinatário realizar a decifragem. O XML Encryption não se preocupa com detalhes de segurança mais amplos, como assinaturas, autenticações e autorizações.

Os dados a serem cifrados podem ser de qualquer tipo: um documento XML, um elemento XML ou o conteúdo de um elemento XML. O resultado da cifragem é um elemento XML EncryptedData que contém (através de um elemento filho) ou identifica (através de uma referência URI) o dado cifrado, mais as informações necessárias para realizar a descifragem.

Quando a cifragem é realizada sobre um elemento XML ou sobre o conteúdo do elemento, o elemento EncryptedData substitui o elemento ou o conteúdo (respectivamente) na versão cifrada do documento XML.

Se a cifragem for realizada sobre dados arbitrários (como um documento XML), o elemento EncryptedData pode se tornar o elemento raiz em um novo documento XML ou ser um elemento filho em um documento XML.

A Figura 3.1 apresenta, de forma resumida, a estrutura do elemento EncryptedData. O símbolo '?' indica elementos opcionais, e o '*' indica que o elemento pode ser repetido várias vezes.

```
<EncryptedData Id? Type?>
  <EncryptionMethod/>?
  <ds:KeyInfo>
    <EncryptedKey>?
    <AgreementMethod>?
    <ds:KeyName>?
    <ds:RetrievalMethod>?
    <ds:*>?
  </ds:KeyInfo>?
  <CipherData>
    <CipherValue>?
    <CipherReference URI?>?
  </CipherData>
</EncryptedData>
```

Figura 3.1: Estrutura do Elemento *EncryptedData*

O elemento CipherData contém ou referencia o dado cifrado. No primeiro caso, o elemento CipherValue contém o dado cifrado; no segundo caso, o atributo URI do elemento CipherReference indica a localização do dado cifrado.

O elemento EncryptionMethod descreve o algoritmo de criptografia aplicado ao dado cifrado. KeyInfo agrupa elementos com informações sobre as chaves de criptografia. EncryptedKey é utilizado para transportar a chave de criptografia do remetente para um destinatário conhecido. AgreementMethod indica o método utilizado para derivação da chave secreta baseada no segredo (chave) compartilhado. KeyName identifica a chave necessária para decifrar o conteúdo do elemento CipherData. RetrievalMethod é utilizado para indicar a localização da chave de criptografia.

3.3. XML Signature

XML Signatures [BAR01] são assinaturas digitais projetadas para uso em mensagens XML. O padrão define um esquema de representação para o resultado de uma operação de assinatura digital aplicada a dados arbitrários (freqüentemente XML). A XML Signature provê autenticação, integridade de dados e suporte à não-repudiação dos dados assinados.

Uma característica fundamental da XML Signature é a habilidade de assinar somente partes específicas de um documento XML ao invés de assinar todo o documento. Isso é interessante quando um documento tem uma longa história em que diferentes partes do documento foram criadas em diferentes tempos por diferentes autores, cada um assinando os elementos que criou. Essa flexibilidade também é útil quando se tem um documento onde apenas algumas partes podem ser modificadas.

A validação da assinatura requer que os dados assinados estejam acessíveis. A assinatura irá geralmente indicar a localização dos dados originais. Eles podem:

- Ser referenciados por uma URI dentro da assinatura;
- Residir na mesma mensagem XML em que está a assinatura;
- Estar embutidos dentro da assinatura;
- Conter a assinatura dentro da mensagem (a assinatura está embutida).

A Figura 3.2 apresenta a estrutura de uma assinatura do XML Signature. O símbolo '?' indica elementos opcionais, e o '*' indica que o elemento pode ser repetido várias vezes.

```
<Signature>
   <SignedInfo>
      (método de canonização)
      (método de assinatura)
      (<Reference (URI=)?>
         (Transforms)?
         (DigestMethod)
         (DigestValue)
      </Reference>)+
   </SignedInfo>
   (SignatureValue)
   (KeyInfo)?
   (Object)*
</Signature>
```

Figura 3.2: Estrutura do XML Signature

SignedInfo contém as informações sobre o dado que está sendo assinado. Reference contém informações sobre o método utilizado para fazer o *hash* (DigestMethod) e o valor obtido na sua aplicação (DigestValue). Pode opcionalmente conter as informações sobre as transformações realizadas sobre o dado antes da aplicação do *hash* (elemento Transforms). SignatureValue contém a assinatura dos dados. KeyInfo indica a chave a ser utilizada para validar a assinatura. Object, quando utilizado, pode conter qualquer tipo de dado.

3.4. IPsec

O IPsec [KEN98a] [OPP98] é uma plataforma aberta formada por um conjunto de protocolos que provêem serviços de autenticação, integridade, controle de acesso e confidencialidade da camada de rede IP, tanto em ambientes IPv4 como em ambientes IPv6. O IPsec é uma das opções para implementação de VPNs e seus serviços podem ser utilizados por quaisquer protocolos de camadas superiores, como TCP, UDP, HTTP, SMTP e outros.

Além de ser um padrão aberto IETF que está sendo adotado por muitos fabricantes de equipamentos de redes de computadores e desenvolvedores de sistemas, o IPsec possui uma arquitetura aberta que possibilita a inclusão de novos algoritmos de autenticação e criptografia.

3.4.1. Arquitetura do IPsec

O protocolo IPsec opera em um *gateway* ou em um computador, com os requisitos de segurança descritos por um banco de dados de política de segurança (SPD – *Security Policy Database*) mantido pelo usuário, pelo administrador de rede ou por uma aplicação

operando dentro de limites pré-definidos. Pode ser utilizado para proteger uma ou mais conexões entre um par de computadores, entre dois *gateways* ou entre um computador e um *gateway*.

Os pacotes IP são selecionados através de três formas de processamento definidas por seletores. Os seletores comparam o pacote IP e as informações da camada de transporte com as entradas do banco de dados SPD para determinar qual ação deve ser tomada. A ação pode ser submeter o pacote aos serviços IPsec, aceitar o pacote sem utilizar os serviços ou ainda descartar o pacote.

A arquitetura básica IPsec é formada pelos protocolos AH [KEN98b] (*Autentication Header*), ESP [KEN98c] (*Encapsulating Security Payload*) e IPComp [SHA01] (*IP Payload Compression*). O primeiro fornece integridade e autenticação da origem dos dados, além do serviço *anti-replay*[6]. O segundo fornece confidencialidade (criptografia) e as mesmas funções do protocolo AH. Os dois protocolos podem ser aplicados conjuntamente ou independentemente um do outro, dependendo do tipo de segurança desejada. O terceiro protocolo permite realizar a compressão dos pacotes IP. Também pode ser aplicado em conjunto com o AH e/ou ESP, ou isoladamente. Os processos de criptografia, autenticação e ou compressão são chamados de transformações.

Cada um destes protocolos suporta dois modos de operação: modo transporte e modo túnel. No modo transporte, os protocolos provêem proteção primária aos protocolos das camadas superiores, e no modo túnel os protocolos de nível superior são colocados dentro de um envelope AH ou ESP.

Além desses dois protocolos, há ainda o protocolo IKE [HAR98] (*Internet Key Exchange*), que é utilizado para estabelecer as associações de segurança (SA – *Security Association*) e para fazer a geração de chaves. O SA é uma conexão lógica unidirecional que define os serviços de segurança, portanto, para uma conexão bidirecional, são necessários dois SAs, um para cada sentido.

O IKE originalmente era chamado de ISAKMP/Oakley, por ser derivado desses dois protocolos. O ISAKMP [MAU98] fornece um *framework* para estabelecer associações de segurança e chaves criptográficas. Esse *framework* é independente de tecnologia e pode ser utilizado com qualquer mecanismo de segurança disponível. Como o ISAKMP não define um

---

[6] Evita que uma mensagem capturada durante uma comunicação legítima seja retransmitida novamente com um propósito ilegítimo.

mecanismo de segurança, essa tarefa é realizada pelo protocolo Oakley [ORM98], que define o protocolo para troca de chaves dentro do ISAKMP.

A Figura 3.3 apresenta a arquitetura do IPsec.

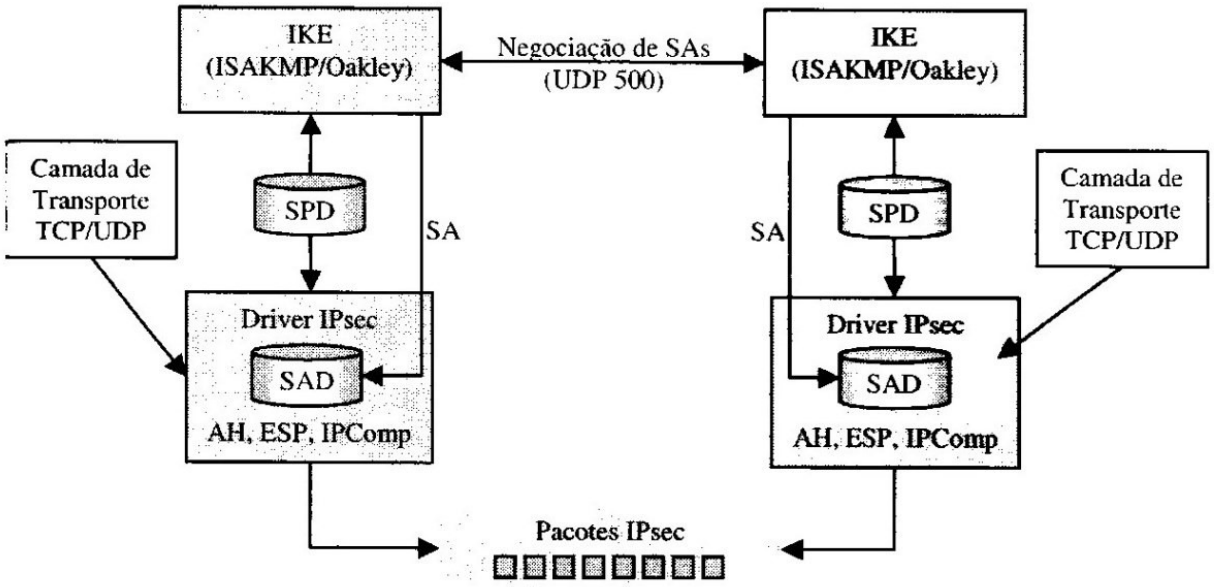

Figura 3.3: Arquitetura do IPsec

O IPsec mantém duas bases de dados relacionadas às SAs:

- Security Policy Database (SPD): base de dados de políticas, que especifica quais serviços de segurança podem ser oferecidos para o tráfego IP, dependendo de fatores como origem, destino, protocolo, etc. Contém uma lista ordenada de políticas separadas em tráfego de entrada e/ou saída. Essas políticas podem especificar que determinado tráfego poderá passar sem precisar de processamento IPsec, determinado tráfego seja descartado e o resto processado pelo IPsec. As entradas nessa base de dados são similares às regras dos firewalls ou filtros de pacotes.
- *Security Associations Database* (SAD): base de dados de associações de segurança, que contém as informações sobre cada SA, como algoritmos e chaves do AH ou ESP, números de seqüência, o modo utilizado e o tempo de vida das SAs. Para o tráfego de saída, uma entrada no SPD aponta para uma entrada no SAD. Isso é, o SPD determina qual SA deve ser utilizada para um determinado pacote. Para o tráfego de entrada, o SAD é consultado para determinar como o pacote deve ser processado.

Detalhes de implementação do protocolo IPsec podem ser encontrados em [KER97].

3.4.2. Protocolos IPsec

IP Authentication Header (AH)

O protocolo AH [KEN98b] é utilizado para prover integridade e autenticação aos pacotes IP e é efetivo contra ataques de IP *spoofing* e de roubo de sessão. A autenticação é feita em todos os campos possíveis, inclusive algumas partes do cabeçalho. Somente os campos que podem ser modificados durante a transmissão do pacote não são incluídos na autenticação (são chamados campos mutáveis), pois não é possível prever os valores na recepção. Os campos são os seguintes:

- Type of Service (TOS);
- Flags;
- Fragment Offset;
- Time to Live (TTL);
- Header Checksum.

Nos casos em que é necessário proteger esses campos, deve-se utilizar o modo túnel. Todo o pacote é considerado não modificável e está sempre protegido pelo AH.

Uma outra funcionalidade do AH é a proteção contra reenvio de pacotes antigos, de utilização opcional. Embora seja opcional, este serviço deve obrigatoriamente ser implementado por qualquer sistema IPsec.

Os seguintes algoritmos de autenticação devem ser implementados obrigatoriamente por um sistema IPsec para o AH:

- HMAC-MD5-96 (RFC 2403) [MAD98a];
- HMAC-SHA-1-96 (RFC 2404) [MAD98b].

Encapsulating Security Payload (ESP)

O protocolo ESP [KEN98c] é utilizado para prover verificação de integridade, autenticação e cifragem para pacotes IP. Assim como no AH, há opcionalmente a proteção contra reenvio de pacotes antigos.

O conjunto desejado de serviços é selecionado no momento da criação da SA. Entretanto, existem algumas restrições:

- A verificação de integridade e a autenticação não podem ser utilizadas separadamente;
- Proteção contra reenvio da mesma mensagem só pode ser ativada com verificação de integridade e autenticação;
- A proteção contra reenvio de pacotes antigos só pode ser selecionada pelo receptor;

Além dessas restrições, é recomendável utilizar a verificação de integridade e autenticação quando a criptografia for utilizada. Isso porque utilizar somente a criptografia pode permitir que intrusos forjem pacotes para tentar um ataque de cripto-análise. Como opção, pode-se utilizar o protocolo AH para realizar a autenticação, enquanto o protocolo ESP realiza a criptografia.

Apesar da criptografia e da autenticação (com integridade) serem opcionais, pelo menos uma delas deve ser sempre selecionada. Caso contrário não faz sentido utilizar o ESP. Se ambas criptografia e autenticação estiverem ativadas, então o receptor primeiro faz a autenticação do pacote e somente se esse passo tiver sucesso executa-se a decifragem. Esse modo de operação diminui a utilização de recursos computacionais e reduz a possibilidade de ataques de negação de serviço.

Os seguintes algoritmos de criptografia devem ser implementados obrigatoriamente por um sistema IPsec para o ESP:

- DES-CBC (RFC 2405) [MAD98c];
- NULL (RFC 2410) [GLE98].

Além desses algoritmos, podem ser implementados ainda (opcionalmente):

- CAST-128 (RFC 2451) [PER98b];
- RC5 (RFC 2451) [PER98b];
- IDEA (RFC 2451) [PER98b];
- Blowfish (RFC 2451) [PER98b];
- 3DES (RFC 2451) [PER98b].

Os seguintes algoritmos de autenticação devem ser implementados obrigatoriamente por um sistema IPsec para o ESP:

- HMAC-MD5-96 (RFC 2403) [MAD98a];
- HMAC-SHA-1-96 (RFC 2404) [MAD98b];
- NULL (RFC 2410) [GLE98].

IP Payload Compression (IPComp)

O *IP payload compression protocol* [SHA01] (protocolo de compressão de dados IP) é um protocolo para redução do tamanho dos pacotes IP. O objetivo é diminuir o tamanho dos pacotes transmitidos e aumentar o desempenho da comunicação entre dois computadores.

Cada pacote IP é comprimido e descomprimido individualmente, ou seja, não tem relação com outros pacotes, pois os pacotes podem chegar fora de ordem ou nem chegar no destino. Cada pacote comprimido encapsula um único pacote IP.

A compressão de pacotes deve ser realizada antes de qualquer outro processamento de segurança, como criptografia e autenticação, e antes de qualquer fragmentação. Se o ganho da compressão for menor que o tamanho do cabeçalho IPComp, o pacote deve ser enviado sem ser comprimido.

Os seguintes algoritmos podem ser implementados por um sistema IPsec para o IPComp (são opcionais):

- LZS (RFC 2395) [FRI98];
- Deflate (RFC 2394) [PER98a];
- OUI – algoritmo de compressão proprietário – deve ser acompanhado por um atributo que identifica o fornecedor específico do algoritmo.

Modo Transporte e Modo Túnel

No modo transporte, o cabeçalho IPsec é acrescentado logo após o cabeçalho IP. No caso do ESP, o *trailer* e a autenticação opcional são acrescentados ao final do pacote original. Se o pacote já tiver um cabeçalho IPsec, então o novo cabeçalho será inserido antes deste. A Figura 3.4 apresenta os pacotes IP com os protocolos AH e ESP no modo transporte.

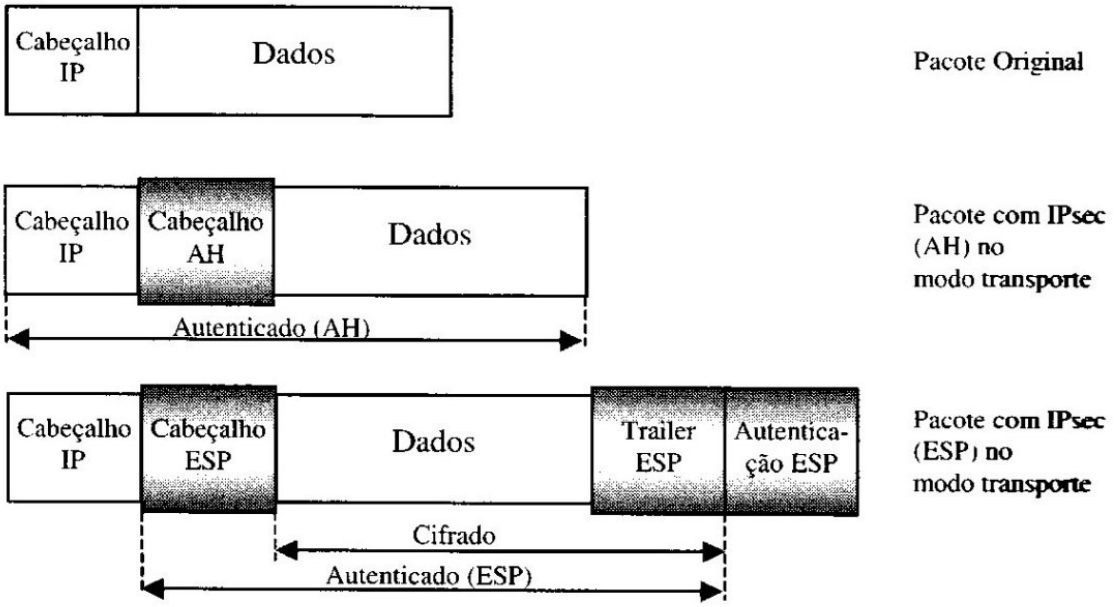

Figura 3.4: IPsec no Modo Transporte

O modo transporte é utilizado normalmente por computadores. Os *gateways* não têm a obrigação de suportar o modo transporte.

A vantagem da utilização do modo transporte é a menor sobrecarga de processamento. Uma desvantagem é que os campos mutáveis não são autenticados. O ESP no modo transporte não provê nem autenticação nem criptografia do cabeçalho IP. Se o AH não for utilizado junto com o ESP, pacotes falsos podem ser entregues para o processamento ESP (um ataque de *spoof*). Outra desvantagem do modo transporte é que os endereços de origem e destino dos pacotes podem ser vistos. Isso pode ser um problema nos casos em que são utilizados IP privados ou que a estrutura interna de endereçamento não deva ser visível na Internet.

No modo túnel, é aplicado o conceito de encapsulamento, ou seja, um novo pacote IP é construído e o pacote original é inserido como dado nesse novo pacote. Então o modo transporte é aplicado ao novo pacote. No caso do ESP, o pacote original torna-se dado no novo pacote, portanto a proteção é total se a criptografia e a autenticação estiverem ativas. Entretanto, o cabeçalho do novo pacote somente será autenticado quando for utilizada a autenticação do AH. A Figura 3.5 apresenta os pacotes IP com os protocolos AH e ESP no modo túnel.

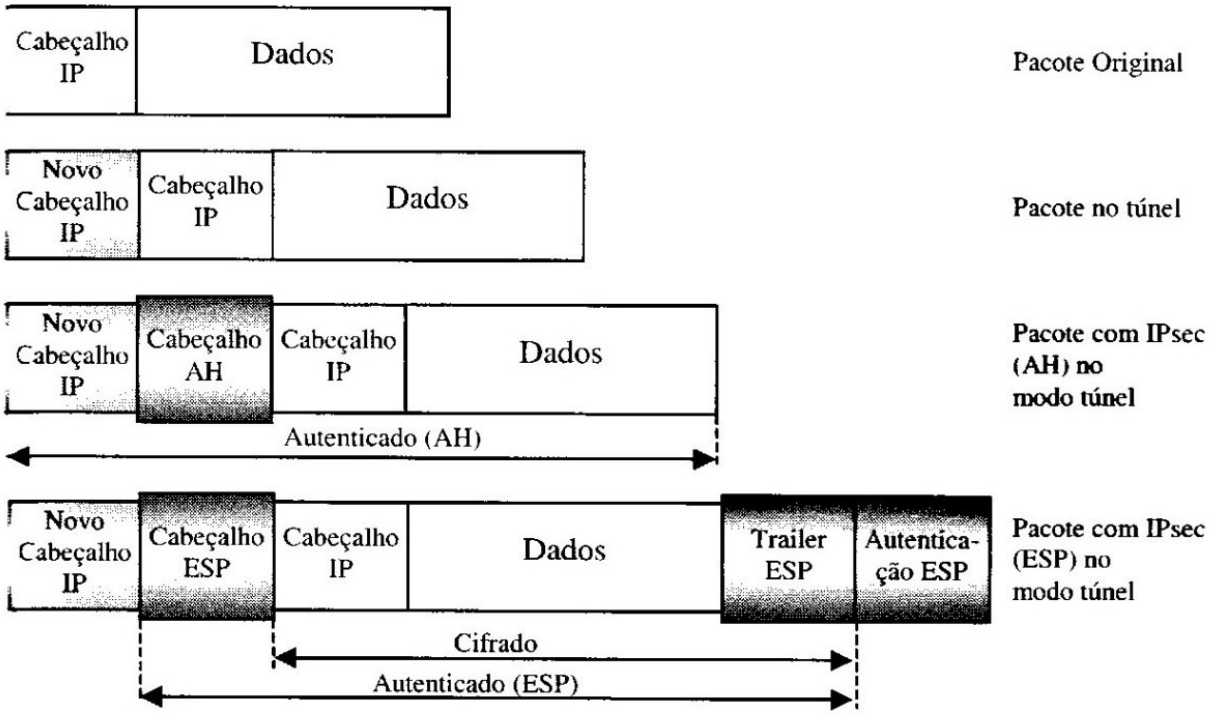

Figura 3.5: IPsec no Modo Túnel

O modo túnel deve ser utilizado sempre que os dois extremos forem *gateways*. Assim, todo o trafego entre os *gateways* estará protegido e as redes internas não poderão ser conhecidas.

Embora se suponha que os *gateways* suportem somente o modo túnel, em alguns casos pode ser necessário que eles suportem também o modo transporte. Esse modo é utilizado quando os *gateways* agem como computadores, como nos casos em que o tráfego é destinado a eles mesmos (por exemplo tráfego de comandos SNMP[7] e ICMP[8]).

No modo túnel os novos cabeçalhos não precisam ter o mesmo endereço dos cabeçalhos originais. Por exemplo, dois *gateways* podem operar um túnel AH em que é utilizada a autenticação de todo o tráfego entre as redes, sendo que os endereços origem e destino do pacote original são de dois computadores, e os do novo pacote são os *gateways*. Esse é um modo típico de operação.

As vantagens do modo túnel são a total proteção do pacote IP encapsulado e a possibilidade de utilizar endereços privados. Entretanto, é necessária uma carga de processamento maior nesse modo.

---

[7] Simple Network Management Protocol
[8] Internet Control Message Protocol

3.4.3. Negociação de Requisitos de Segurança

Associações de Segurança - SAs

Uma SA (*Security Association*) é uma conexão lógica unidirecional entre dois sistemas IPsec. Um SA agrupa todas as informações necessárias para execução de uma comunicação IPsec. A SA especifica, entre outras informações:

- Qual protocolo está sendo utilizado (AH, ESP ou IPComp);
- O algoritmo de autenticação (quando necessário) e as chaves necessárias por esse algoritmo;
- O algoritmo de criptografia (quando necessário) e as chaves necessárias por esse algoritmo;
- O algoritmo de compressão (quando necessário);
- A freqüência de troca das chaves;
- Os tempos de vida das chaves;
- O tempo de vida da própria SA;

Pelo fato das SAs serem unidirecionais, para uma comunicação bidirecional entre dois sistemas IPsec é necessário definir duas SAs, uma para cada sentido. Uma SA define os serviços de segurança para os protocolos AH, ESP ou IPComp, mas não os três ao mesmo tempo. Isto é, para uma conexão que deve ser protegida tanto pelo AH quanto pelo ESP, é necessário definir duas SAs para cada sentido.

O SPI (*Security Parameter Index*) é um número que identifica unicamente uma SA. Um dispositivo, quando recebe um pacote IPsec, extrai o SPI do pacote e procura em sua lista qual SA é necessária para processar este pacote. Então o pacote é autenticado, decifrado e/ou descomprimido de acordo com as regras do SA, utilizando os algoritmos e chaves pré-negociados.

Internet Key Exchange Protocol – IKE

O objetivo do *Internet Key Exchange Protocol* – IKE – é negociar os requisitos de segurança de forma que as duas partes concordem com os algoritmos e os parâmetros, além de realizar a troca de chaves. Em outras palavras, o protocolo estabelece e mantém as SAs utilizadas pelos protocolos AH, ESP e IPComp.

O IKE combina o protocolo ISAKMP desenvolvido pela *US National Security Agency* e o protocolo Oakley desenvolvido pela *University of Arizona*. O ISAKMP é utilizado

para negociar algoritmos e estruturas matemáticas para o Diffie-Hellman [RES99] e para o passo de autenticação. O Oakley é utilizado para realizar a troca de chaves.

O protocolo IKE possui duas fases. A primeira fase estabelece a chave secreta principal a partir do qual todas as chaves criptográficas serão derivadas para proteção do tráfego de dados. No caso mais geral, a criptografia de chave pública [YOU01] é utilizada para estabelecer a SA entre os sistemas para o ISAKMP. A fase 1 somente estabelece a proteção para as mensagens ISAKMP, mas não estabelece SAs e chaves para proteção de dados do usuário.

Na fase 1, as operações utilizam mais o processador, mas são realizadas com pouca freqüência. Uma única execução da fase 1 pode suportar múltiplas execuções da fase 2. Na segunda fase as operações são menos complexas, pois elas são utilizadas somente após a proteção negociada na primeira fase estar ativa. O objetivo da fase 2 é negociar o conjunto de SAs e chaves que irá proteger os dados dos usuários. As mensagens da fase 2 são protegidas pelas SAs geradas na fase 1. Geralmente a fase 2 ocorre mais freqüentemente que a fase 1. Enquanto a fase 1 pode ser executada uma vez por dia ou até uma vez por semana, a fase 2 pode ser executada a cada 2 ou 3 minutos.

O protocolo IKE possui três modos de operação para troca de chaves e para negociação das SAs – dois modos para a fase um e um modo para a fase dois:

- Modo principal (*main mode*): executado na fase um para estabelecimento do canal seguro. Esse modo utiliza seis mensagens para a negociação.
- Modo agressivo (*agressive mode*): é uma outra forma de execução da fase um, um pouco mais simples e mais rápida que o modo principal, mas não provê proteção de identidade para os dispositivos, pois as identidades são transmitidas antes do canal seguro ser negociado. A Implementação desse modo é opcional. Nesse modo, apenas três mensagens são trocadas entre os dispositivos durante a negociação.
- Modo rápido (*quick mode*): executado na fase dois para negociação de uma SA para as comunicações.

Para estabelecer uma SA, o dispositivo que inicia a negociação propõe seis itens:

- Os algoritmos de criptografia;

- Os algoritmos de *hash*[9];
- O método de autenticação;
- Informações sobre o grupo Diffie-Hellman [RES99];
- Uma função pseudo-aleatória (PRF – *pseudo-random function*) utilizada para gerar um *hash* de certos valores durante a troca de chaves para propósitos de verificação (é opcional);
- O tipo de proteção a ser utilizado (AH ou ESP) e, quando for o caso, o protocolo de compressão (IPComp).

O domínio de interpretação do IPsec (IPsec DOI) [PIP98] define quais são os protocolos de segurança e algoritmos de criptografia disponíveis e padroniza a nomenclatura destes.

Combinação de SAs

Os protocolos AH e ESP podem ser aplicados sozinhos ou em combinação. Dados os dois modos de cada protocolo, há um número de combinações possíveis. Ainda é possível configurar os dois protocolos de forma que não tenham origens e/ou destinos idênticos.

São possíveis diversas combinações de SAs. Alguns exemplos são:

- Ambos os protocolos são aplicados no modo transporte sobre o mesmo pacote IP (Figura 3.6). No envio, os pacotes IP são cifrados pelo ESP e após são autenticados pelo AH, e na recepção a autenticação deverá ser realizada antes da decifragem. Essa seqüência deve ser seguida para evitar que o sistema precise decifrar (grande carga de processamento) pacotes que falhem na autenticação.

[9] HASH: função que executa uma equação matemática sobre um determinado texto e cria um código chamado *message digest* (resumo da mensagem). A equação matemática deve ser de tal maneira que seja impossível obter o texto a partir do resultado e que seja muito difícil que dois textos gerem o mesmo valor *hash*.

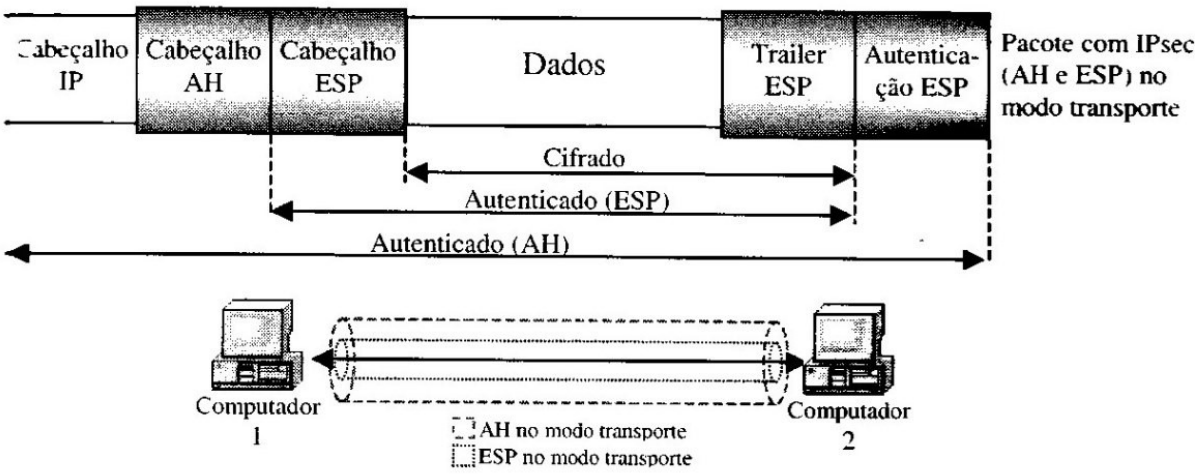

Figura 3.6: IPsec com AH e ESP no Modo Transporte

- Tunelamento aninhado: os protocolos são aplicados no modo túnel em seqüência (Figura 3.7): Após cada aplicação um novo cabeçalho IP é criado e o próximo protocolo é aplicado. Um exemplo é o caso de duas redes conectadas por dois *gateways*. Os *gateways* fazem a autenticação dos pacotes transportados entre as redes e os computadores origem e destino fazem a criptografia dos dados. Outra opção é aplicar o modo transporte entre dois computadores de redes diferentes e o modo túnel entre os *gateways*.

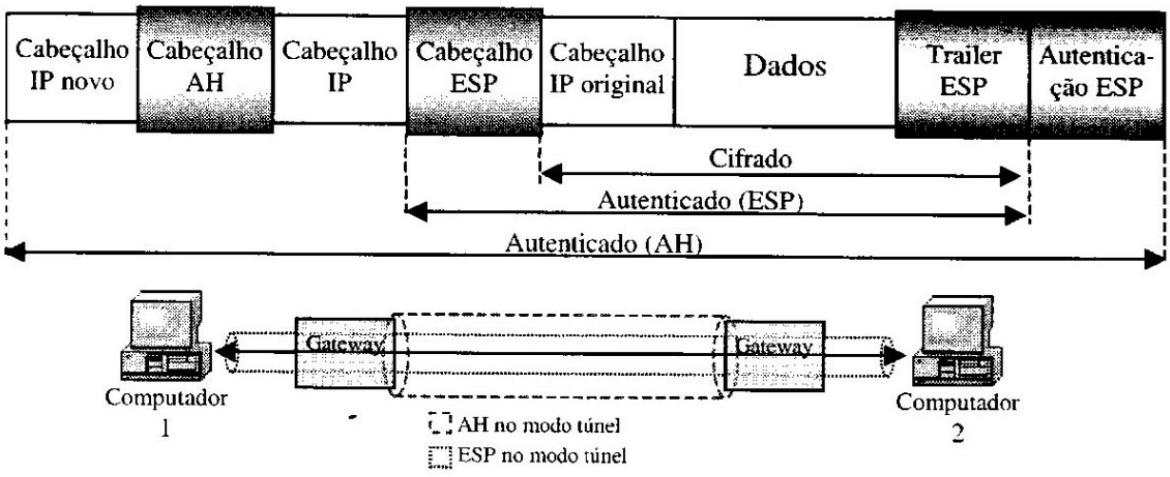

Figura 3.7: IPsec com AH no Modo túnel e ESP no Modo túnel

3.5. SSL/TLS

O SSL (*Secure Sockets Layer*) é um protocolo utilizado para proteger comunicações na Internet, implementado na camada de aplicação. Foi desenvolvido originalmente pela Netscape, e foi aceito universalmente para autenticação e criptografia de dados entre clientes e servidores. O novo padrão IETF chamado TLS (*Transport Layer Security*) é baseado no SSL.

As principais características do SSL são:

- Segurança em conexões cliente/servidor: o SSL garante o sigilo dos dados trocados entre as partes envolvidas na conexão através do uso de criptografia simétrica. A fim de evitar que as mensagens, mesmo decifradas, sejam modificadas, o SSL adiciona às mensagens um MAC (*Message Authentication Code*). Além de sigilo e integridade, o SSL ainda provê a autenticação das partes envolvidas a fim de garantir e verificar a identidade das mesmas. Neste processo, o SSL utiliza criptografia assimétrica e certificados digitais;
- Interoperabilidade: dada a sua especificação bem detalhada e o uso de algoritmos criptográficos conhecidos, diferentes implementações do protocolo têm a garantia de interagir entre si;
- Extensibilidade: permite que novos parâmetros e métodos de criptografia (assimétrica ou simétrica) sejam incorporados ao protocolo;
- Eficiência: devido à demanda por recursos computacionais que este tipo de operação requer, o protocolo dispõe da opção de armazenar em memória *cache* as informações referentes às sessões, diminuindo desta forma o esforço computacional em sucessivas conexões.

O protocolo SSL está localizado acima da camada TCP/IP e abaixo de protocolos de mais alto nível como o HTTP, o IMAP e outros. Tipicamente a implementação deste protocolo é feita na própria aplicação. Ele permite que servidores que o implementem possam se autenticar aos clientes que também o implementem, e vice-versa.

O SSL inclui dois sub-protocolos: o SSL *record protocol* e o SSL *handshake protocol*. O primeiro é responsável por encapsular os dados de camadas superiores em pacotes compactados e cifrados e repassá-los para a camada de transporte. O segundo utiliza o primeiro para que o servidor e o cliente possam se autenticar e negociar os algoritmos de

criptografia e as chaves criptográficas antes que o protocolo de aplicação receba ou envie os dados.

Quando um cliente SSL inicia uma conexão com um servidor SSL, é necessário que ambos utilizem a mesma versão do protocolo e negociem os algoritmos de criptografia a serem utilizados. A autenticação, realizada através de certificados digitais, é opcional, sendo que geralmente somente o servidor é autenticado.

3.6. Object Oriented Security

[XAV01] apresenta um modelo de segurança que permite representar a autorização ou bloqueio ao acesso a elementos de um documento XML. Essa representação pode ser realizada indicando diretamente elementos de um determinado documento ou através do esquema desse documento (representado por um DTD[10] [BRA00]).

Uma autoridade central utiliza um conjunto de esquemas para especificar o formato da informação que trafega dentro da organização. Os documentos XML são instâncias dos esquemas definidos e mantidos em cada setor, descrevendo as informações específicas deste setor.

A relação entre a instância e o esquema permite uma distinção entre dois níveis de autorização, ambos permitindo uma alta granularidade nas especificações. O primeiro nível (associado aos documentos XML) permite controlar autorizações de baixo nível, provendo total controle sobre as autorizações de cada documento (são chamadas de autorizações *hard*, pois têm uma prioridade maior que as autorizações do segundo nível). O segundo nível (associado ao DTD) são autorizações de alto nível e provêem controle de autorizações para todos os documentos da organização ou departamento (são chamadas de autorizações *soft*, pois têm prioridade menor).

As autorizações para o esquema e para o documento XML podem ser especificadas referenciando cada elemento/atributo único em um documento. As autorizações de um elemento podem ser declaradas recursivamente (aplicam-se também aos sub-elementos) ou locais (aplicam-se somente aos atributos diretos).

As autorizações especificadas para os esquemas ou documentos XML são armazenados em documentos chamados XML *Access Sheet* (XAS). As autorizações são

---

[10] DTD - *Document Type Definition* – Utilizado para definir a estrutura de documentos XML. Atualmente está sendo substituído pelo XML Schema [FAL01].

expressas em XML. Cada autorização indica se há permissão ou proibição para um usuário executar uma certa ação sobre um objeto, junto com a prioridade (*soft* ou *hard*) e o tipo (recursivo ou local). O objeto identifica um elemento ou um conjunto de elementos em um documento ou conjunto de documentos.

Os documentos e esquemas são caracterizados por URIs (*Uniform Resource Identifier*). Como existe alta granularidade, é necessário referenciar elementos e atributos específicos dentro dos documentos. Esses elementos e atributos são referenciados pela linguagem XPath [CLA99] (linguagem para representar expressões de caminhos).

Os usuários e as solicitações de acesso são representados através da identificação do usuário e/ou a localização de onde se originam as solicitações, onde as localizações podem ser expressas através do endereço IP ou através de nomes simbólicos (p. ex. jose.infy.com). Os usuários são caracterizados pela tripla <identificação do usuário, endereço IP, nome simbólico máquina>, onde a identificação do usuário é o nome utilizado no *login*, o endereço IP é o endereço da máquina de onde o usuário fez a solicitação e o nome simbólico é o nome da máquina registrado no servidor DNS.

As autorizações podem ser especificadas através de grupos de usuários ou padrões de localização. Os grupos de usuário são os grupos definidos no servidor. O padrão de localização é uma expressão identificando um conjunto físico de localizações, referenciando o nome simbólico (*.edu representa todas as máquinas do domínio educacional) ou o endereço IP (192.168.* representa todas as máquinas da sub-rede 192.168).

O processamento utiliza a linguagem XSL [ADL01] e pode ser realizado tanto no servidor quanto no cliente. O cenário principal envolve um usuário solicitando um conjunto de elementos XML de um computador remoto. O processador usa como entrada o documento XML válido solicitado pelo usuário, junto com o documento XAS associado. O processador também usa o esquema do documento e o XAS associado. A saída do processador é um documento XML válido incluindo somente as informações às quais o usuário tem permissão de acesso. Esse documento é então enviado ao usuário como resultado da solicitação.

expressas em XML. Cada autorização indica se há permissão ou proibição para um usuário executar uma certa ação sobre um objeto, junto com a prioridade (*soft* ou *hard*) e o tipo (recursivo ou local). O objeto identifica um elemento ou um conjunto de elementos em um documento ou conjunto de documentos.

Os documentos e esquemas são caracterizados por URIs (*Uniform Resource Identifier*). Como existe alta granularidade, é necessário referenciar elementos e atributos específicos dentro dos documentos. Esses elementos e atributos são referenciados pela linguagem XPath [CLA99] (linguagem para representar expressões de caminhos).

Os usuários e as solicitações de acesso são representados através da identificação do usuário e/ou a localização de onde se originam as solicitações, onde as localizações podem ser expressas através do endereço IP ou através de nomes simbólicos (p. ex. jose.infy.com). Os usuários são caracterizados pela tripla <identificação do usuário, endereço IP, nome simbólico máquina>, onde a identificação do usuário é o nome utilizado no *login*, o endereço IP é o endereço da máquina de onde o usuário fez a solicitação e o nome simbólico é o nome da máquina registrado no servidor DNS.

As autorizações podem ser especificadas através de grupos de usuários ou padrões de localização. Os grupos de usuário são os grupos definidos no servidor. O padrão de localização é uma expressão identificando um conjunto físico de localizações, referenciando o nome simbólico (*.edu representa todas as máquinas do domínio educacional) ou o endereço IP (192.168.* representa todas as máquinas da sub-rede 192.168).

O processamento utiliza a linguagem XSL [ADL01] e pode ser realizado tanto no servidor quanto no cliente. O cenário principal envolve um usuário solicitando um conjunto de elementos XML de um computador remoto. O processador usa como entrada o documento XML válido solicitado pelo usuário, junto com o documento XAS associado. O processador também usa o esquema do documento e o XAS associado. A saída do processador é um documento XML válido incluindo somente as informações às quais o usuário tem permissão de acesso. Esse documento é então enviado ao usuário como resultado da solicitação.

3.7. XML Security Services Suite (XS-Cube)

[TAK02] descreve um sistema de segurança que permite a sua implantação sem a necessidade de realizar modificações nos sistemas existentes. O sistema consiste de um servidor *proxy*[11] e um servidor de assinatura/verificação.

O servidor *proxy* (ver Figura 3.8) é colocado entre cada aplicação e a Internet para examinar as mensagens XML trocadas entre as aplicações. Esse servidor chama um serviço de segurança no servidor de assinatura/verificação de acordo com o conteúdo da mensagem. Quando a mensagem está sendo enviada, o servidor de assinatura/verificação assina a mensagem XML para que então essa possa ser enviada para o destinatário. Quando a mensagem está sendo recebida, o servidor de assinatura/verificação do destinatário verifica se a assinatura está correta, a assinatura é removida e a mensagem é encaminhada para a aplicação.

O servidor de assinatura pode ainda prover formas de seleção de chaves de assinatura de acordo com o conteúdo do documento, utilizando regras de seleção no formato XML. Além disso, os documentos assinados podem ser armazenados em um banco de dados.

Atualmente o XS-Cube implementa somente a assinatura de mensagens. Estão sendo incluídas as funcionalidades de criptografia, gerenciamento de chaves e controle de acesso. O servidor de assinatura é uma das funcionalidades do XS-Cube.

[11] O *Proxy* atua como um intermediário entre a rede local e a Internet, ou seja, de um lado recebe requisições de clientes e as repassa para o servidor na internet e do outro lado recebe as requisições da internet e as repassa para o servidor. O *Proxy* se faz passar pelo cliente ou pelo servidor, dependendo de qual lado ele seja implementado.

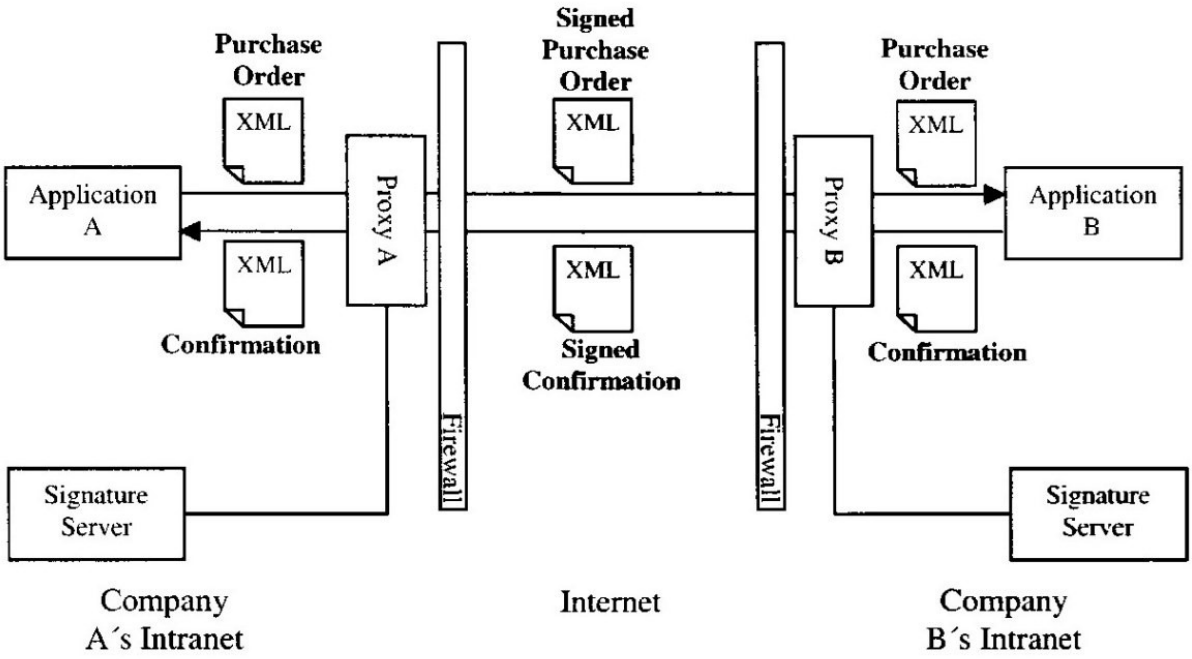

Figura 3.8: Arquitetura XS-Cube

O funcionamento é o seguinte:

- A aplicação A na companhia A envia um documento de pedido para a aplicação na companhia B.
- Como o documento de pedido passa pelo *proxy* A, o documento é enviado para o servidor de assinatura e é assinado antes de ser enviado para a companhia B.
- Na companhia B, o *proxy* B recebe o documento assinado e o envia para o servidor de verificação. O servidor de verificação verifica a assinatura do documento.
- O resultado da verificação é retornado para o *proxy* B. Se a assinatura é válida, o *proxy* B remove a assinatura do documento e envia o documento para a aplicação B. É possível manter informações sobre a assinatura se for necessário para aplicação.
- A aplicação B recebe o documento. Depois de processar, a aplicação B gera o documento de resposta e envia para a companhia A.

- Como o documento de resposta passa pelo *proxy* B, o documento é enviado para o servidor de assinatura aonde é assinado e então enviado para a companhia A.
- Na companhia A, o *proxy* A redireciona a mensagem para o servidor de verificação. Se a assinatura é válida, a assinatura é removida do documento e envia o documento para aplicação A.

Como o *proxy* verifica as mensagens XML na rede e decide quando devem ser assinadas ou verificadas, as aplicações A e B não precisam ter conhecimento sobre assinatura e verificação. Portanto, aplicações existentes não precisam ser modificadas.

O servidor de assinatura/verificação é implementado na forma de um *Web Service* que aceita objetos a serem assinados ou verificados. Pelo fato de utilizar mensagens SOAP, os serviços podem ser disponibilizados para vários protocolos de transporte. Todos os documentos assinados e verificados são armazenados em um banco de dados. O sistema utiliza o XML *Access Control Language* (XACL) [HAD00] para controlar o acesso, de forma que é possível controlar quais usuários têm permissão de acesso a cada elemento do documento XML.

3.8. Outros Mecanismos

Além dos mecanismos apresentados, existem outros mecanismos que também propõem a segurança para transmissão de mensagens. Abaixo são descritos sucintamente alguns desses mecanismos:

- Flexible Low-Cost Internet Extended-Enterprise (FloCI-EE) [FUR02b]: combina controle de acesso baseado em papéis (RBAC[12]), infra-estrutura de chave pública (PKI[13]) e infra-estrutura de gerenciamento de privilégios (PMI[14]) para criar um mecanismo de segurança para *Web Services* que provê autenticação e autorização em comunicações entre organizações.
- WS-Security [ATK02]: define um mecanismo que inclui integridade, confidencialidade e autenticação de mensagens dentro de mensagens SOAP. Esses mecanismos podem ser utilizados independentemente ou em combinação. O WS-Security utiliza as especificações XML Signature e

[12] RBAC: Role-Based Acess Control
[13] PKI: Public-Key Infrastructure
[14] PMI: Privilege Management Infrastructure

XML Encryption e define como incluir assinaturas digitais, *hash's* e representação dos dados cifrados na mensagem SOAP.

3.9. Conclusão do Capítulo

Esse capítulo apresentou alguns mecanismos de segurança que podem ser utilizados para proteção de mensagens. Alguns dos mecanismos fazem a proteção de cada mensagem individualmente ou parte da mensagem, enquanto outros criam um canal seguro por onde trafegam as mensagens.

Entre os mecanismos que protegem as mensagens, estão o XML Encryption, XML Security e o XS-Cube. Os dois primeiros exigem que sejam feitas modificações no sistema que for utilizá-los, pois as mensagens deverão estar protegidas no momento do envio. Uma possibilidade é incluir uma camada que faça a proteção, como proposto pelo projeto XS-Cube. Mas isso também não é simples, pois é necessário equipamento e programas adicionais para realizarem todas as operações necessárias.

Outra possibilidade é a utilização de mecanismos de proteção que estabelecem comunicações seguras implementadas na pilha de protocolos de rede. Entre eles pode-se citar o SSL e o IPsec, que são os mais comuns. O SSL é normalmente implementado pela aplicação, portanto também tem a desvantagem de demandar modificações na aplicação existente para permitir a sua utilização. Já o IPsec é implementado na camada de rede, portanto é transparente para aplicações e pode ser utilizado sem necessidade de realizar modificações em aplicações existentes.

Para organizações que desejam disponibilizar *Web Services* pela Internet, o uso do IPsec é bastante indicado, pois freqüentemente essas organizações possuem *Firewalls* como equipamento de segurança e muitos desses equipamentos já possuem uma implementação interna do IPsec. Ou seja, não é necessário modificar as implementações dos *Web Services*; basta realizar a configuração do *Firewall* para permitir a criação de canais IPsec.

Apesar de existirem críticas contra o IPsec [FER00], ele pode ser visto como um mecanismo de segurança eficiente para transporte de informações pela Internet [TSC98], além de ser um protocolo que é mandatório para implementações do IPv6 [DEE98].

Capítulo 4

Modelos IETF para Representação de Políticas IPsec

4.1. Introdução ao Capítulo

A configuração de políticas em um computador ou equipamento pode ser realizada diretamente, através de uma aplicação desenvolvida com esse objetivo. Essa aplicação armazena as informações em uma estrutura, que pode ser eventualmente proprietária e diferente de outras aplicações, desde que a implementação do IPsec nesse equipamento seja capaz de interpretá-la.

O problema surge quando as políticas devem ser compartilhadas entre vários equipamentos, sendo que podem existir diferentes sistemas operacionais (no caso de computadores) ou diferentes marcas de equipamentos (em equipamentos como *firewall*), cada um com uma implementação diferente do IPsec. Para permitir a distribuição das políticas de forma que todos possam interpretá-la, é necessário adotar um padrão.

O IETF está trabalhando no desenvolvimento de um modelo para representação de políticas IPsec [JAS02]. Esse modelo é apresentado na forma de diagramas UML, definindo classes, seus atributos e as relações com outras classes necessárias para a representação das políticas IPsec.

O presente capítulo apresenta os modelos existentes para representação de políticas IPsec, que foram utilizadas como base para o desenvolvimento deste trabalho. A seção 4.2 apresenta um modelo genérico para representação de informações e a seção 4.3 apresenta uma especialização deste modelo para representação genérica de políticas. A seção 4.4 descreve um modelo específico para representação do IPsec, com o detalhamento das classes e seus atributos. Na seção 4.5 é apresentada uma política utilizando esse modelo de

representação. Os modelos apresentados nesse capítulo são representados através da notação UML (*Unified Modeling Language*) [UML03].

4.2. O Modelo de Informações CIM

O CIM (*Common Information Model*) [WES00] desenvolvido pelo DMTF (*Distributed Management Task Force*)[15] é um modelo de informação orientado a objetos, destinado à representação de elementos gerenciados. O CIM foi projetado para receber informações de agentes SNMP, CMIP[16], DMI[17] e outros. Permite que aplicações corporativas de diferentes desenvolvedores que utilizam plataformas heterogêneas descrevam, criem e compartilhem objetos de gerência. A intenção é criar aplicações de gerência corporativa e ferramentas que possam monitorar e controlar todas as redes, sistemas e aplicações existentes nas organizações.

Os esquemas de gerenciamento são a base para plataformas e aplicações de gerenciamento, como configuração de dispositivos e gerenciamento de desempenho. O modelo CIM é estruturado de forma que o ambiente gerenciado possa ser visto como uma coleção de sistemas inter-relacionados, cada um dos quais composto de um número de elementos discretos.

O modelo CIM é estruturado em três camadas distintas:

- *Core Model*: um modelo de informações que captura noções aplicáveis a todas as áreas de gerenciamento. É um pequeno conjunto de classes, associações e propriedades que representam o vocabulário básico para análise e descrição de sistemas gerenciados.
- *Common Model*: um modelo de informações que captura noções que são comuns a áreas particulares de gerenciamento, mas independente de tecnologias e implementações. Alguns exemplos de áreas comuns são: sistemas, aplicações, redes, dispositivos e políticas. O modelo de informações é específico o suficiente para prover a base para desenvolvimento de aplicações de gerenciamento. Esse esquema provê um conjunto de classes base para ser estendido pela área de esquemas que

[15] www.dmtf.org
[16] Common Management Information Protocol
[17] Desktop Management Interface

especificam a tecnologia. O *Core Model* e o *Common Model* são chamados de CIM *Schema*.

- *Extension Schemas*: representam as extensões que especificam as tecnologias do *Common Model*. Esses esquemas são específicos para os ambientes, como por exemplo sistemas operacionais (por exemplo, UNIX ou MS Windows).

A habilidade de trocar informações entre aplicações de gerenciamento é fundamental para o CIM. O mecanismo atual para troca de informações é o *Management Object Format* (MOF). No momento, não há interfaces de programação ou protocolos definidos pela especificação do CIM. Portanto, um sistema preparado para utilizar o CIM deve ser capaz de importar e exportar adequadamente construções bem formadas do MOF. Cabe ao sistema definir como as operações de importação e exportação são executadas.

4.3. Os Modelos de Informação PCIM e PCIMe

O PCIM (Policy Core Information Model) [MOO01] é um modelo orientado a objetos que permite a representação de informações sobre políticas. As classes que compõem esse modelo têm a intenção de servir como uma hierarquia de classes extensível, permitindo que desenvolvedores, administradores de rede e administradores de políticas representem informações sobre políticas de diferentes tipos.

São definidas duas hierarquias de classes de objetos: classes estruturais, que definem a forma de representar e controlar a informação de política; e classes associativas, que indicam como as instâncias das classes estruturais se relacionam. Essas classes e associações são suficientemente genéricas para representar políticas de qualquer natureza.

Uma política PCIM consiste de um conjunto de regras (representadas pela classe *PolicyRule*), sendo que cada regra consiste de um conjunto de condições (classe *PolicyCondition*) e um conjunto de ações (classe *PolicyAction*). Se o conjunto de condições associado a uma regra for validado como verdadeiro, então o conjunto de ações associado à mesma regra será executado. Há ainda a classe *PolicyTimePeriodCondition* que permite indicar os períodos de tempo em que a regra está ativa.

A classe *Policy* é a base da hierarquia do PCIM. A classe *PolicyGroup* permite que *PolicyRules* ou *PolicyGroups* sejam agrupados, conforme pode ser visto na Figura 4.1. Por exemplo, é possível criar conjuntos de regras específicas para cada setor dentro de uma

organização, através da criação de um *PolicyGroup* para cada setor. Para determinar quais políticas se aplicam a um determinado setor basta obter o *PolicyGroup* relacionado a ele.

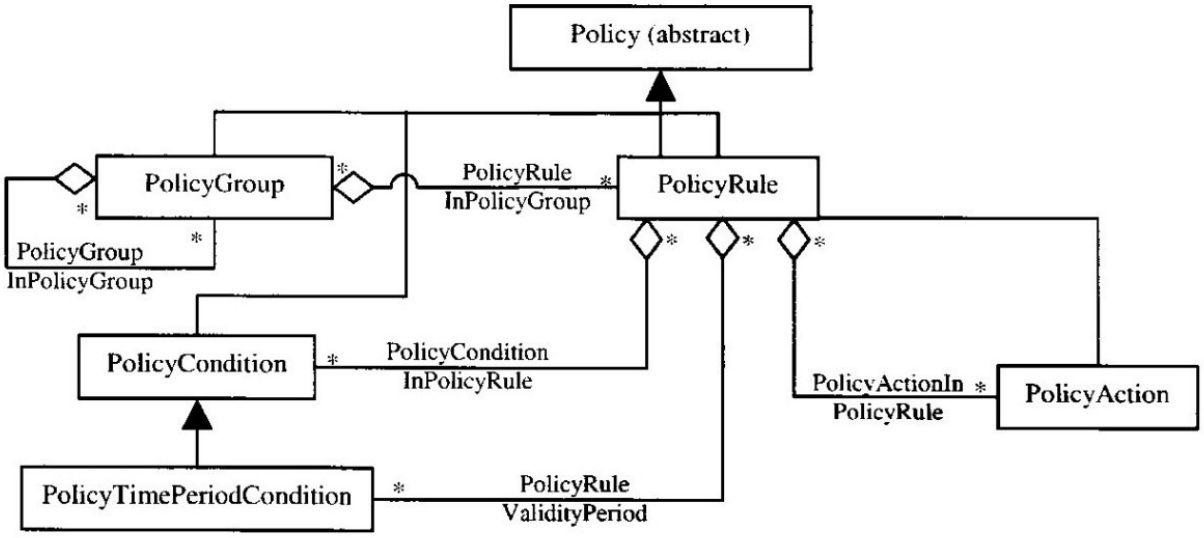

Figura 4.1: Principais Classes do PCIM

A especificação prevê a priorização de regras de políticas. Isso permite definir a ordem em que as regras devem ser aplicadas.

Um exemplo de aplicação em que é necessário utilizar prioridades é quando é necessário expressar exceções dentro de um caso geral. Por exemplo, suponha uma organização em que as pessoas do setor de engenharia não possuem direito de acesso à Internet. Uma pessoa deste mesmo setor necessita o acesso à Internet para realizar pesquisas sobre novas tecnologias. Para cada uma das afirmações anteriores será criada uma regra, mas que serão conflitantes pois a pessoa com direito a acesso pertence ao setor que não tem direito ao acesso. Atribuindo uma prioridade maior para a segunda regra (a exceção) comparada com a primeira regra (o caso geral), o administrador de políticas obtém o efeito desejado: permitir que somente uma pessoa dentro de um grupo obtenha o acesso necessário.

As condições associadas às regras podem ser construídas de duas maneiras:

- Forma normal disjuntiva: um conjunto de condições construído com o operador lógico AND e unido pelo operador lógico OR. Essa forma também é conhecida como *Disjunctive Normal Form* (DNF);

- Forma normal conjuntiva: um conjunto de condições construído com o operador lógico OR e unido pelo operador lógico AND. Essa forma também é conhecida como *Conjunctive Normal Form* (CNF).

A forma de agrupamento é definida através do atributo *ConditionListType* da classe *PolicyRule*. Além disso, as condições podem ser negadas individualmente através do atributo *ConditionNegated* e agrupadas através da atribuição de um valor numérico a cada condição (atributo *GroupNumber*), onde aquelas que possuem valores numéricos iguais são consideradas pertencentes ao mesmo grupo de condições. Para ilustrar as duas formas de agrupamento, suponha um conjunto de cinco condições na forma C_i(*GroupNumber*, *ConditionNegated*), sendo as condições do conjunto: C={C_1(1,*falso*), C_2(1,*verdadeiro*), C_3(1,*falso*), C_4(2,*verdadeiro*), C_5(2,*falso*)}. Portanto, as condições serão avaliadas da seguinte forma:

Se *ConditionListType* = DNF então: avaliação(C)=(C_1 Λ !C_2 Λ C_3) V (!C_4 Λ C_5)[18]

Se *ConditionListType* = CNF então: avaliação(C)=(C_1 V !C_2 V C_3) Λ (!C_4 V C_5)

O PCIMe (*Policy Core Information Model Extensions*) [MOO03] propõe diversas modificações no modelo PCIM. Essas modificações têm o objetivo de solucionar as deficiências encontradas após a publicação do PCIM. Por exemplo, uma modificação foi a inclusão da classe *PolicySet* como uma nova opção para aninhar regras de política e também a inclusão de novas classes para suportar uma seqüência ordenada de ações. Apesar das modificações, o PCIMe ainda mantém a compatibilidade com as implementações já existentes.

4.4. Modelo de Políticas IPsec

O modelo de políticas IPsec [JAS02] é uma das extensões do modelo CIM para representação de segurança de tráfego em redes utilizando IPsec. Esse modelo é derivado do CIM *Core Policy Model* e é fortemente influenciado pelo CIM *Policy Core Information Model* (PCIM) e pelo *Policy Core Information Model Extensions* (PCIMe).

O modelo de políticas IPsec também utiliza elementos de outros modelos CIM, como classes do modelo *Network* e classes do modelo *User*.

---

[18] O símbolo Λ significa conjunção (operação lógica AND) e o símbolo V significa disjunção (operação lógica OR).

O propósito principal do modelo IPsec é permitir a representação e configuração dos protocolos IKE e IPsec. O modelo define um *framework* para especificar configurações de Associações de Segurança pré-configuradas e/ou controlar a inicialização e configuração de negociações do IKE.

O modelo inclui classes para representação do estado interno, tabelas de conversão de nomes de rede e representação de canais IPsec que devem ser inicializados quando o serviço IPsec for iniciado. Essas classes não serão apresentadas neste trabalho por não fazerem parte do escopo.

Os principais atributos são apresentados neste capítulo, com uma breve descrição. Para maiores detalhes sobre os atributos, consultar o Anexo A.

4.4.1. Classes de Políticas e Regras

O modelo de políticas IPsec é derivado do PCIM. Isso significa que cada política contém um conjunto ordenado de objetos de regra onde cada regra tem uma condição, representada pela classe SACondition, e uma ação, representada pela classe SAAction (Figura 4.2). Quando uma decisão de política deve ser tomada, as regras são avaliadas seqüencialmente de acordo com o seu grupo de política e prioridade. A ação da regra cuja condição seja satisfeita é então executada. O modelo IPsec restringe o uso de SAActions a uma escolha ordenada ao invés de uma lista de ações a serem executadas.

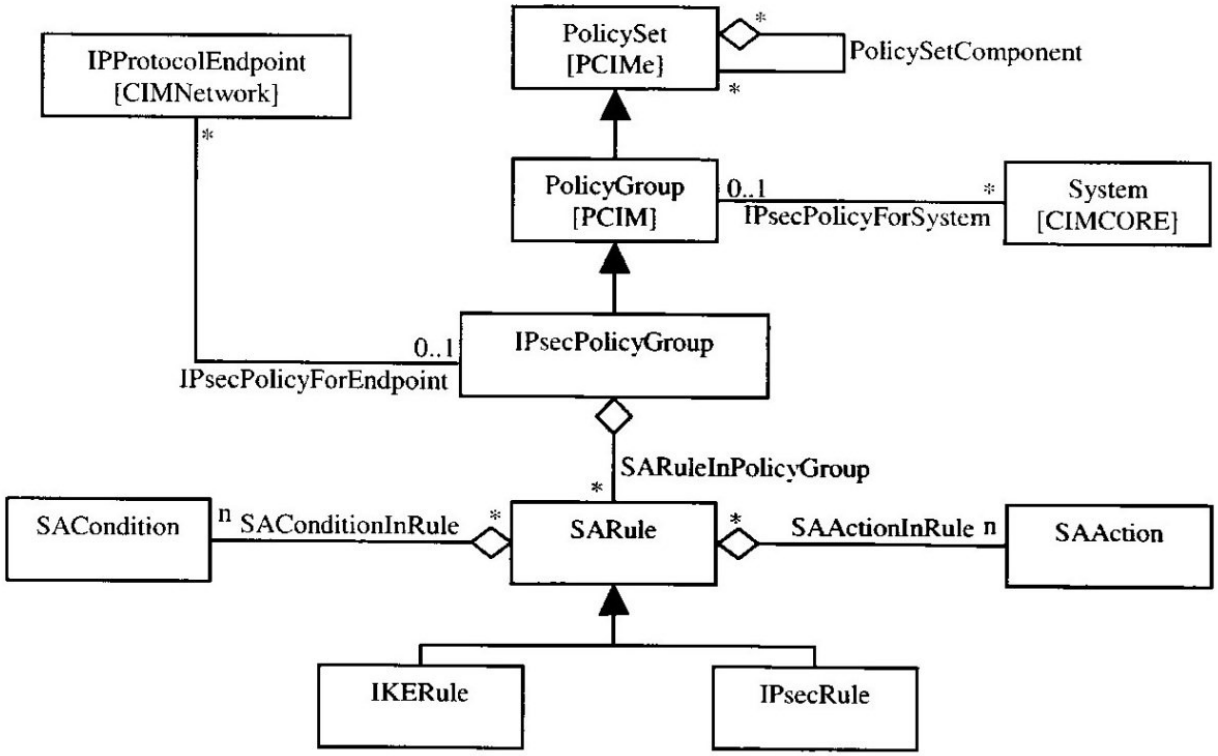

Figura 4.2: Políticas e Regras

As políticas são representadas pela classe IPsecPolicyGroup. Essa classe pode agregar dois tipos de regras: IKERule (para negociações da fase 1 do IKE) e IPsecRule (para negociações da fase 2 do IKE). A classe IPsecPolicyGroup pode agregar instâncias dela mesma, através da agregação PolicySetComponent herdada de PolicySet, permitindo a construção de políticas aninhadas. A classe IPProtocolEndpoint associada à classe IPsecPolicyGroup representa o IP ao qual as políticas se aplicam.

A função principal das classes de regras é mapear condições às ações. As condições são padrões de parâmetros de tráfego IP (endereços, portas e protocolos), credenciais de computadores como certificados digitais ou dados de identidade IKE. Quando algum evento ocorre para disparar a avaliação das políticas (como um tráfego desprotegido chegando na interface de rede), as características desse evento são utilizadas para testar as condições das regras e encontrar a primeira compatível. A ação da primeira regra é então utilizada para preencher os parâmetros da negociação IKE (algoritmos de criptografia aceitáveis, tempo de vida de SAs, etc.).

À medida que a negociação progride são obtidas informações adicionais. Por exemplo, no modo principal do IKE é necessário concluir as propostas da fase 1 antes de obter a identificação autenticada do computador. Portanto, as regras da fase 1 que têm

condições de endereço e identidade são utilizadas quando as informações se tornam disponíveis.

Na Tabela 4.1 são apresentadas as descrições das classes estruturais e os respectivos atributos e na Tabela 4.2 são apresentadas as descrições das classes associativas.

Tabela 4.1: Descrição das classes de Políticas e Regras

Classe	Descrição e Atributos
IPsecPolicyGroup	Essa classe serve como container para classes IPsecPolicyGroup ou para um conjunto de classes SARules. Principais atributos: • PolicyDecisionStrategy: Método de execução das ações na política. Esse atributo indica se são executadas somente as ações da primeira regra encontrada ou ações de todas as regras. • PolicyRoles: é uma abstração normalmente utilizada para identificar a interface dos dispositivos onde se deseja aplicar a política.
IPProtocolEndpoint	Representa uma determinada interface em um computador ou equipamento, que possui um endereço IP. Principais atributos: • Address: endereço IP que a classe IPProtocolEndpoint representa. • SubnetMask: máscara de rede do endereço IP. • AddressType: indica o tipo de endereçamento utilizado (IPv4, IPv6 ou desconhecido).
PolicySet	Classe abstrata que permite agrupar políticas dentro de um conjunto estruturado de políticas.

Tabela 4.1: Descrição das classes de Políticas e Regras (cont.)

Classe	Descrição e Atributos
SARule	Serve como classe base para as classes IKERule e IPsecRule. Apesar de ser uma classe concreta, não deve ser instanciada. Ela define um ponto comum para associações com condições e com ações para os dois tipos de regras. Principais atributos: • ConditionListType: indica se a lista de condições associadas a essa regra deve ser tratada na forma normal disjuntiva (DNF) ou na forma normal conjuntiva (CNF). • ExecutionStrategy: determina o comportamento das ações. Pode ser: executar até o sucesso, executar tudo, executar até que um falhe. • SequencedActions: indica se a seqüência de ações deve ser seguida ou não. Os valores são: obrigatório, recomendado, não importa. • PolicyRoles: é uma abstração normalmente utilizada para identificar a interface dos dispositivos onde se deseja aplicar a política. • PolicyDecisionStrategy: Método de execução das ações na política. Pode ser: ações da primeira regra encontrada, ações de todas as regras.
IKERule	Associa condições e ações para as negociações da fase 1 do IKE. Os atributos são os mesmos da classe SARule.
IPsecRule	Associa condições e ações para as negociações da fase 2 do IKE. Os atributos são os mesmos da classe SARule.

Tabela 4.2: Descrição das classes Associativas das Políticas e Regras

Classe	Descrição e Atributos
SARuleInPolicyGroup	Associa uma regra (SARule) ao grupo de políticas que a contém. Principais atributos: • Priority: contém um inteiro que identifica a prioridade relativa da regra relacionada dentro do grupo de regras da política. Números maiores indicam prioridades maiores.
SAConditionInRule	É uma classe de agregação entre regras (SARule) e condições (SACondition). Principais atributos: • GroupNumber: contém um inteiro que identifica o número do grupo ao qual a condição associada pertence. • ConditionNegated: Indica se a condição associada deve ser negada.
SAActionInRule	É uma classe de agregação entre regras (SARule) e ações (SAAction). Para uma regra IKE (IKERule), a ação deve ser relacionada à fase 1, ou seja, IKEAction ou IKERejectAction. Da mesma forma, se a regra for IPsecRule as ações devem ser relacionadas à fase 2 (p. ex. IPsecTransportAction) ou uma ação pré-configurada (p. ex. IPsecBypassAction). Principais atributos: • ActionOrder: identifica a posição relativa da ação associada dentro da seqüência de ações associadas à classe SARule.
PolicySetComponent	É uma classe de agregação que permite a criação de políticas aninhadas. Principais atributos: • Priority: contém um inteiro que identifica a prioridade relativa da política relacionada dentro do grupo de políticas aninhadas a uma determinada política. Números maiores indicam prioridades maiores.

As regras IKE e IPsec são utilizadas para construir ou para negociar as associações de segurança, que são então armazenados no SAD (*Security Associations Database*). As regras IPsec representam o SPD (*Security Policy Database*).

O uso das regras IKE e IPsec pode ser descrito como:

- Caso um pacote esteja saindo desprotegido, primeiro deve-se verificar as regras IPsec. Se for encontrada alguma regra correspondente, verifica-se o SAD. Se não existir um SA correspondente e se a regra Ipsec exigir uma negociação, a regra IKE correspondente será utilizada. O SA negociado ou pré-configurado será armazenado no SAD.
- Caso um pacote desprotegido esteja entrando no sistema, primeiro verifica-se as regras IPsec. Se for encontrada alguma regra, verifica-se o SAD. Se nenhum SA for encontrado mas existir um SA pré-configurado, então esse SA será armazenado no SAD. Esse comportamento aplica-se somente para as ações de permissão e descarte de pacotes.
- Caso um pacote protegido esteja entrando no sistema, primeiro verifica-se as regras IPsec. Se for encontrada alguma regra, verifica-se o SAD. Caso não seja encontrado um SA correspondente mas exista um SA pré-configurado, então esse SA será armazenado no SAD.
- Se um pacote de negociação IKE estiver entrando no sistema, que não é parte de um SA IKE já existente, ele será verificado contra as regras IKE. A condição para uma regra IKE (SACondition) é usualmente composta por uma instância da classe PeerIDPayloadFilterEntry (tipicamente para uma negociação com o IKE no modo agressivo) ou por uma instância da classe IPHeadersFilter. O SA resultante da negociação será armazenado no SAD.

É esperado que, quando uma negociação IKE esteja para se iniciar, o conjunto de regras IKE seja verificado. A verificação das regras IKE será baseada no pacote IKE usando as propriedades da classe IPHeadersFilter.

4.4.2. Classes de Filtros e Condições

As classes de condições e filtros IPsec (Figura 4.3) são utilizadas para construir a parte da condição das regras IKE e IPsec. As condições são verificadas contra as informações

do cabeçalho IP, da identidade do computador e/ou as credenciais para determinar qual regra deve ser utilizada para a negociação.

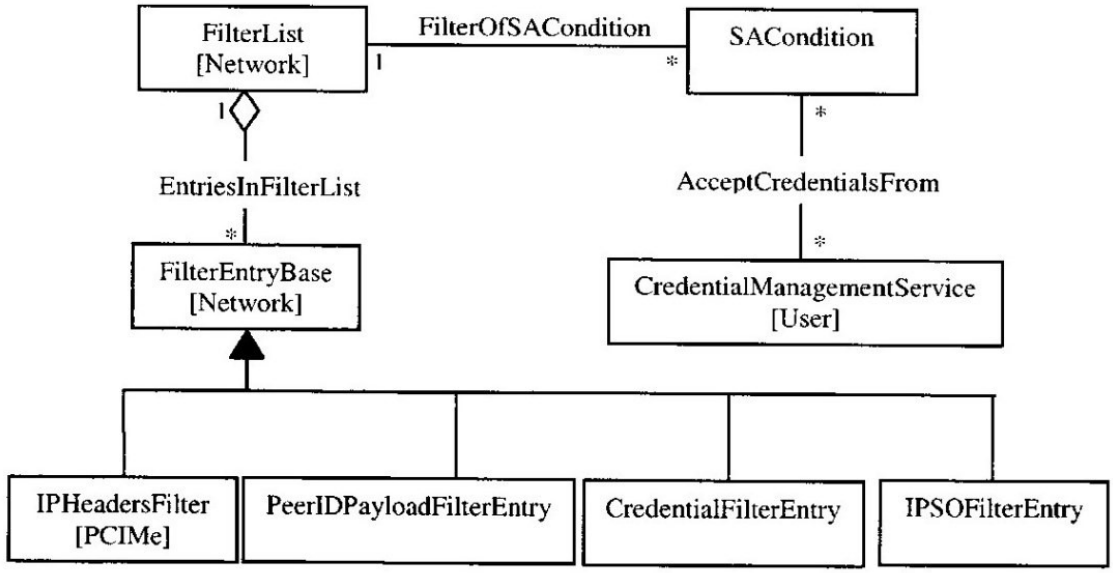

Figura 4.3: Filtros e Condições

Cada instância da classe SACondition está associada a uma instância da classe FilterList, que agrega um conjunto de instâncias de filtros. Os filtros especificam valores de baixo nível que permitem verificar dados como o endereço IP e a porta ou a identidade do computador.

A Tabela 4.3 apresenta as descrições das classes estruturais e os respectivos atributos e na Tabela 4.4 são apresentadas as descrições das classes associativas.

Tabela 4.3: Descrição das Classes de Filtros e Condições

Classe	**Descrição e Atributos**
SACondition	Define condições das regras para negociações do IKE e do IPsec.
CredentialManagementService	Representa uma entidade que gerencia credenciais e seus ciclos de vida. Pode ser uma autoridade certificadora, um servidor Kerberos, entre outros.

Tabela 4.3: Descrição das Classes de Filtros e Condições (cont.)

Classe	Descrição e Atributos
FilterList	Classe que agrega instâncias da classe (ou subclasses de) FilterEntryBase, através da agregação EntriesInFilterList. É possível agregar diferentes tipos de filtros em um único FilterList (é aplicado o operador lógico AND entre os filtros para determinar o resultado do FilterList). Principais atributos: • Direction: indica a direção do fluxo de dados ao qual o filtro será aplicado (pode ser não aplicável, entrada, saída ou ambos).
FilterEntryBase	Classe abstrata que serve como classe base para os filtros. • IsNegated: Indica que resultado da aplicação do filtro deve ser negado.
IPHeadersFilter	Contém as propriedades mais comuns para filtragem de pacotes IP, TCP e UDP. As propriedades que não estiverem presentes são tratadas como "todos os valores". Principais atributos: • HdrSrcAddress e HdrSrcAddressEndOfRange: indicam o(s) endereço(s) de origem. • HdrSrcMask: máscara do endereço de origem. • HdrDestAddress e HdrDestAddressEndOfRange: indicam o(s) endereço(s) de destino. • HdrDestMask: máscara do endereço de destino. • HdrProtocolID: identificador do tipo de protocolo. • HdrSrcPortStart e HdrSrcPortEnd: indicam a(s) porta(s) IP do endereço de origem. • HdrDestPortStart e HdrDestPortEnd: indicam a(s) porta(s) IP do endereço de destino.

Tabela 4.3: Descrição das Classes de Filtros e Condições (cont.)

Classe	Descrição e Atributos
CredentialFilterEntry	Define uma classe que casa credenciais de dispositivos IKE. As credenciais podem ser certificados X.509, tickets Kerberos ou outros tipos de credenciais obtidos durante a fase 1 do IKE. Principais atributos: • CredentialType: tipo de credencial utilizada (X.509 ou tickets Kerberos). • MatchFieldName: especifica um campo da credencial que será verificado. • MatchFieldValue: especifica o valor que será comparado com o campo da credencial.
IPSOFilterEntry	É utilizada para filtrar o tráfego baseado nos valores do cabeçalho IP *Security Options* como definido na RFC 1108 [KEN91]. Esse tipo de filtro é utilizado para ajustar o nível de criptografia do IPsec de acordo com a classificação IPSO do tráfego (secreto, confidencial, restrito, etc.). Principais atributos: • MatchConditionType: indica o campo do cabeçalho que será testado. • MatchConditionValue: indica o valor que o campo indicado por MatchConditionType deve conter.
PeerIDPayloadFilterEntry	Define filtros usados para casar os valores do ID *payload* do protocolo IKE. Permite a especificação de certos valores como "*@company.com" ou "193.190.125.0/24". Esse filtro só é aplicável quando o dispositivo estiver agindo como receptor. Principais atributos: • MatchIdentityType: especifica o tipo de identidade. • MatchIdentityValue: especifica o valor que o campo deve conter.

Tabela 4.4: Descrição das Classes Associativas de Filtros e Condições

Classe	Descrição e Atributos
FilterOfSACondition	Associa a classe SACondition com a classe FilterList, que especifica os filtros desta condição.
EntriesInFilterList	Agrega um conjunto de entradas de filtro (subclasses da classe FilterEntryBase) a uma classe FilterList.
AcceptCredentialsFrom	Associa serviços de gerenciamento de credenciais considerados confiáveis para certificar credenciais (por exemplo: uma autoridade certificadora ou um servidor Kerberos).

4.4.3. Classes de Ações

As classes de ações (Figura 4.4) são utilizadas para modelar as diferentes ações que um dispositivo IPsec pode realizar quando as condições associadas à regra forem satisfeitas.

As ações estáticas são derivadas da classe SAStaticAction. Essas ações não requerem negociação. Por exemplo, IPsecBypassAction é uma classe estática que indica que o tráfego deve ser enviado ou recebido sem processamento IPsec. A classe PreconfiguredSAAction é outro tipo de ação estática que corresponde a associações de segurança pré-configuradas, que não são negociadas. A Tabela 4.5 apresenta as classes com suas respectivas descrições e atributos, e a Tabela 4.6 apresenta as classes associativas.

As ações de negociação especificam alguns parâmetros tanto para a fase 1 (IKEAction) quanto para a fase 2 (IPsecTransportAction ou IPsecTunnelAction). Os demais parâmetros da negociação são encontrados nas instâncias das classes SAProposal.

Para as instâncias da classe IPsecTunnelAction onde o outro computador é um *gateway* de segurança, uma associação com a classe PeerGateway provê informações adicionais sobre esse *gateway*. Nesses casos, após determinar o endereço *gateway*, o mesmo deve ser reaplicado às políticas para determinar se pode ser alcançado diretamente ou se é necessário criar um túnel aninhado.

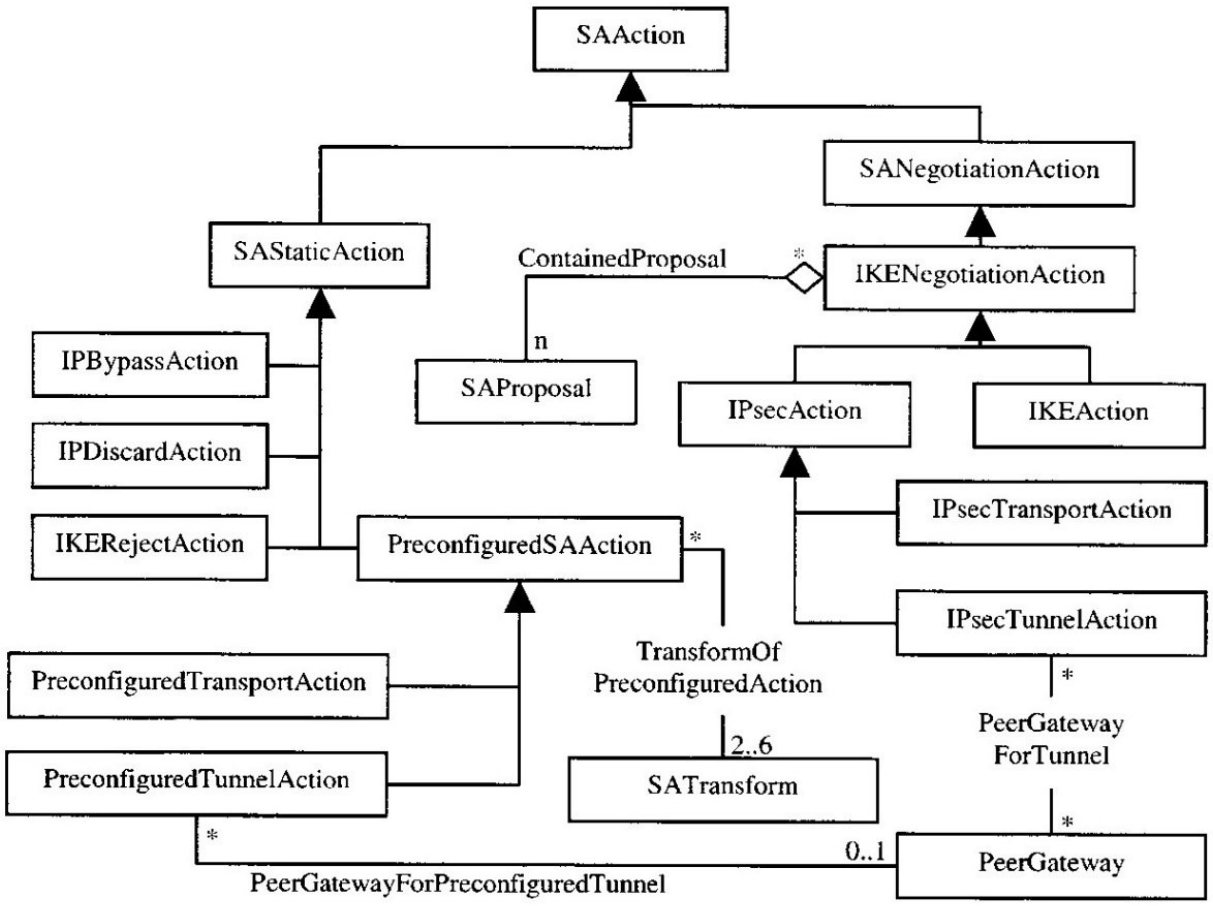

Figura 4.4: Ações

Tabela 4.5: Descrição das Classes de Ações

Classe	Descrição e Atributos
SAAction	Serve como a classe base para as ações do IKE e do IPsec. É usada para diferentes tipos de agregação das ações do IKE e do IPsec. É uma classe abstrata. Principais atributos: • DoActionLogging: indica se deve ser gerado um registro de *log* quando uma ação é executada. • DoPacketLogging: indica se deve ser gerado um registro de *log* quando o SA resultante for utilizado para processar um pacote.
SANegotiationAction	Especifica uma ação solicitando uma negociação das políticas de segurança. É uma classe abstrata.

Tabela 4.5: Descrição das Classes de Ações (cont.)

Classe	Descrição e Atributos
SAStaticAction	Serve como a classe base para ações IKE e IPsec que não requerem qualquer negociação. É uma classe abstrata. Principais atributos: • LifetimeSeconds: tempo de vida da associação de segurança em segundos.
IpsecBypassAction	É utilizada para indicar que é permitido o tráfego de pacotes sem aplicar o IPsec.
IpsecDiscardAction	É utilizado para indicar que os pacotes devem ser descartados. É o mesmo que dizer que os pacotes são negados.
IKERejectAction	É utilizado para prevenir tentativas do IKE de realizar negociações. O principal uso dessa classe é prevenir ataques de negação de serviço quando o IKE estiver atuando como receptor. Isso vai além de simplesmente descartar pacotes UDP da porta 500 pois é possível utilizar a classe PeerIDPayloadFilterEntry para indicar endereços específicos.
IKENegotiationAction	Serve como classe base para as ações do IKE e do IPsec que resultam em negociações IKE. É uma classe abstrata. Principais atributos: • MinLifetimeSeconds: tempo de vida mínimo em segundos que será aceito em uma negociação. É utilizado para prevenir ataques de negação de serviço. • MinLifetimeKilobytes: tempo de vida mínimo em KBytes que será aceito em uma negociação. É utilizado para prevenir ataques de negação de serviço.
IpsecAction	Serve como classe base para as ações IPsec transporte e túnel. Especifica parâmetros usados para a fase 2 da negociação IKE. Principais atributos: • UsePFS: indica se deve ser utilizado o *Perfect Forward Secrecy* (criação de novas chaves não depende das chaves utilizadas anteriormente).

Tabela 4.5: Descrição das Classes de Ações (cont.)

Classe	Descrição e Atributos
IpsecTransportAction	É uma subclasse da classe IPsecAction e é utilizada para especificar o uso da associação de segurança no modo transporte.
IpsecTunnelAction	É uma subclasse da classe IPsecAction e é utilizada para especificar o uso da associação de segurança no modo túnel.
IKEAction	Especifica os parâmetros que são utilizados para a negociação da fase 1 do IKE. Principais atributos: • ExchangeMode: indica o modo utilizado pelo IKE para a negociação.
PeerGateway	Especifica o *gateway* seguro com o qual o serviço IKE negocia. Principais atributos: • Name: especifica o nome do *gateway* de segurança. • PeerIdentityType: especifica o tipo de identificação utilizado (os tipos são definidos no IPsec DOI [PIP98]) • PeerIdentity: é a identificação do *gateway*, do tipo indicado no atributo PeerIdentityType.
PreconfiguredSAAction	É utilizada para criar uma associação de segurança utilizando algoritmos e chaves pré-configurados. Embora essa classe seja concreta, ela não deve ser instanciada. Principais atributos: • LifetimeKilobytes: tempo de vida da associação de segurança em KBytes.
PreconfiguredTransportAction	É utilizada para criar uma associação de segurança no modo transporte, usando algoritmos e chaves pré-configuradas.
PreconfiguredTunnelAction	É utilizada para criar uma associação de segurança no modo túnel, usando algoritmos e chaves pré-configuradas.

Tabela 4.6: Descrição das Classes Associativas de Ações

Classe	Descrição e Atributos
ContainedProposal	É uma classe de agregação que associa um conjunto ordenado de proposições (SAProposal) com uma classe de negociação (IKENegotiationAction). Se a classe de negociação for IKEAction, então as proposições devem ser da classe IKEProposal. Se a classe de negociação for derivada de IPsecAction, então as proposições devem ser da classe IPsecProposal. Principais atributos: • SequenceNumber: número inteiro que indica a ordem de precedência das proposições. Valores menores têm preferência sobre valores maiores.
PeerGatewayForTunnel	É uma classe de associação que liga um conjunto ordenado de classes Gateway à classes IPsecTunnelAction. Principais atributos: • SequenceNumber: número inteiro que indica a ordem de precedência dos *gateways*. Valores menores têm preferência sobre valores maiores.
PeerGatewayForPreconfiguredTunnel	Associa zero ou um PeerGateway com múltiplos PreconfiguredTunnelActions.
TransformOfPreconfiguredAction	Associa uma PreconfiguredSAAction com duas, quatro ou seis SATransforms que serão aplicadas ao tráfego de entrada e saída. Principais atributos: • SPI: Especifica o número do SPI a ser utilizado pela ação pré-configurada para a transformação associada. • Direction: Indica se a propriedade SPI é utilizada para o tráfego de entrada ou de saída.

4.4.4. Classes de Propostas e Transformações

As classes de propostas e transformações (Figura 4.5) modelam as configurações a serem propostas pelo dispositivo IPsec durante a negociação das fases 1 e 2. Uma instância da classe IKEAction agrega um conjunto de instâncias da classe IKEProposal que contém

todas as informações necessárias para compor um conjunto de propostas, ou se estiver recebendo propostas, pode verificar a lista para encontrar se alguma pode ser aceita. Para uma ação IPsec (IPsecTransportAction ou IPsecTunnelAction), cada proposta corresponde a um objeto IPsecProposal.

As propostas IPsec são mais complexas que as propostas IKE porque cada instância da classe IpsecProposal pode especificar múltiplas alternativas para cada tipo de transformação (AH, ESP e IPComp).

A Tabela 4.7 apresenta as descrições das classes estruturais e os respectivos atributos e na Tabela 4.8 são apresentadas as descrições das classes associativas.

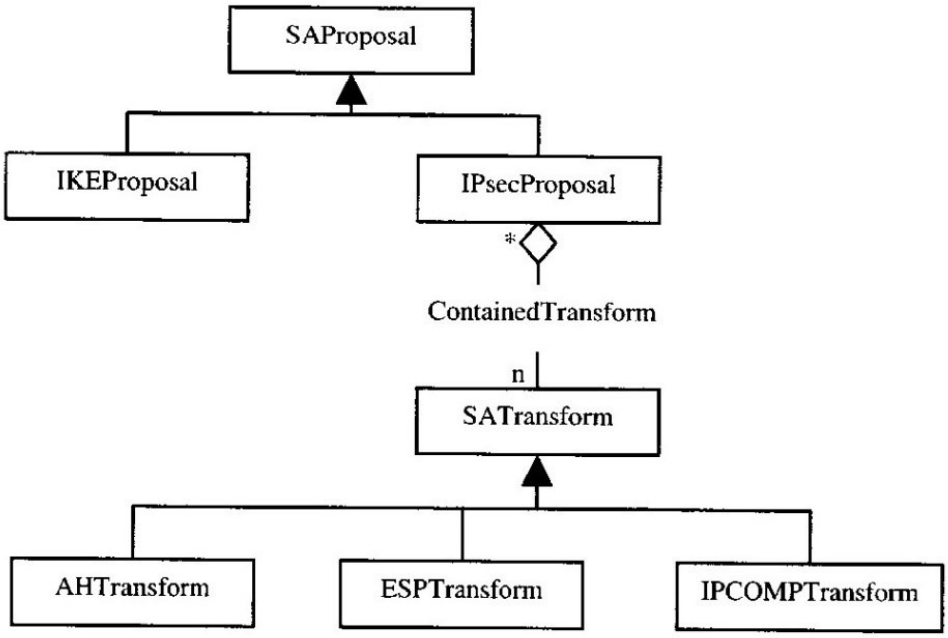

Figura 4.5: Propostas e Transformações

Tabela 4.7: Descrição das Classes de Propostas e Transformações

Classe	Descrição e Atributos
SAProposal	Serve como classe base para as classes de proposta do IKE e do IPsec. Especifica os parâmetros que são comuns para os dois tipos de proposta. É uma classe abstrata.

Tabela 4.7: Descrição das Classes de Propostas e Transformações (cont.)

Classe	Descrição e Atributos
IKEProposal	Especifica os parâmetros propostos necessários para direcionar a negociação da associação de segurança do IKE (fase 1). Principais atributos: • CipherAlgorithm: especifica o algoritmo de criptografia para a fase 1 do IKE. • HashAlgorithm: especifica o algoritmo de *hash* para a fase 1. • PRFAlgorithm: especifica a função pseudo-aleatória para a fase 1. • GroupId: especifica o grupo de troca de chaves das associações de segurança. • AuthenticationMethod: especifica o método de autenticação para a fase 1.
IPsecProposal	Serve como classe base para as classes de proposta do IPsec. Agrega as transformações necessárias para construir a proposta IPsec.
SATransform	Serve como classe base para as transformações IPsec que podem ser utilizadas para compor uma proposta IPsec ou serem usadas como ações pré-configuradas.
AHTransform	Especifica os parâmetros para o algoritmo AH a ser proposto durante a negociação da associação de segurança do IPsec. Principais atributos: • AHTransformId: especifica o identificador do algoritmo de autenticação para o AH. • UseReplayPrevention: indica se deve ser utilizada a proteção contra reenvio da mesma mensagem.

Tabela 4.7: Descrição das Classes de Propostas e Transformações (cont.)

Classe	Descrição e Atributos
ESPTransform	Especifica os parâmetros para o algoritmo ESP a ser proposto durante a negociação da associação de segurança do IPsec. Principais atributos: • IntegrityTransformId: especifica o identificador do algoritmo de integridade. • CipherTransformId: especifica o identificador do algoritmo de criptografia. • CipherKeyLength: tamanho em bits da chave de criptografia. • UseReplayPrevention: indica se deve ser utilizada a proteção contra reenvio de mensagens.
IPCOMPTransform	Especifica o algoritmo IPComp (IP compression) para ser proposto durante a negociação da associação de segurança do IPsec. Principais atributos: • Algorithm: especifica o identificador do algoritmo de compressão proposto. • DictionarySize: especifica o tamanho do dicionário para o algoritmo de compressão. Para algoritmo com tamanho pré-definido, esse valor é ignorado.

Tabela 4.8: Descrição das Classes Associativas de Propostas e Transformações

Classe	Descrição e Atributos
ContainedTransform	É uma classe de agregação que associa uma proposta (IPsecProposal) com um conjunto de transformações (SATransform) que compõe a proposta. Existindo múltiplas propostas do mesmo tipo, deve-se interpretar como uma disjunção entre elas (operação lógica OR). Para propostas de tipos diferentes, deve-se interpretar como uma conjunção (operação lógica AND). Principais atributos: • SequenceNumber: número inteiro que indica a ordem de precedência das transformações. Valores menores têm preferência sobre valores maiores.

4.5. Exemplos de política IPsec

A Figura 4.6 apresenta um exemplo de política IPsec representada utilizando o modelo do IETF.

Para este exemplo, suponha que a política do servidor permita somente o tráfego de dados se for protegido utilizando o IPsec e para os serviços HTTP (porta 80) e SMTP (porta 25). Qualquer outro tráfego será bloqueado.

Para representar essa política, foram utilizadas uma instância da classe IPsecPolicyGroup (com o nome PolíticaExemplo), uma instância da classe IKERule e uma da classe IPsecRule (essas duas indicando quais fluxos são permitidos e suas respectivas ações de proteção) e mais uma instância da classe IKERule para indicar que os demais fluxos devem ser bloqueados (chamada Regra3). Por questões de espaço somente a duas primeiras regras foram representadas completamente.

A Regra1 contém duas condições, uma contém o filtro para representar o fluxo HTTP com destino à porta 80 do servidor e outra contém o filtro para representar o fluxo SMTP com destino à porta 25 do servidor. A Regra2 possui também duas condições, que são as mesmas da Regra1 (pode-se observar a reutilização de condições).

Se as condições da Regra1 forem satisfeitas, então a ação IKEAction será executada. Da mesma forma, a regra dois também será satisfeita, portanto a ação IPsecAction

será executada A instância da classe IKEAction possui uma proposta, que consiste nos seguintes algoritmos[19]:

- Algoritmo de criptografia: DES-CBC
- Algoritmo de hash: MD5
- Método de autenticação: Assinaturas digitais RSA

A instância da classe IPsecAction possui uma proposta que inclui os protocolos AH e ESP, sendo que:

- Algoritmo de transformação para o AH: MD5
- Algoritmo de integridade do ESP: SHA-1
- Algoritmo de criptografia do ESP: 3DES

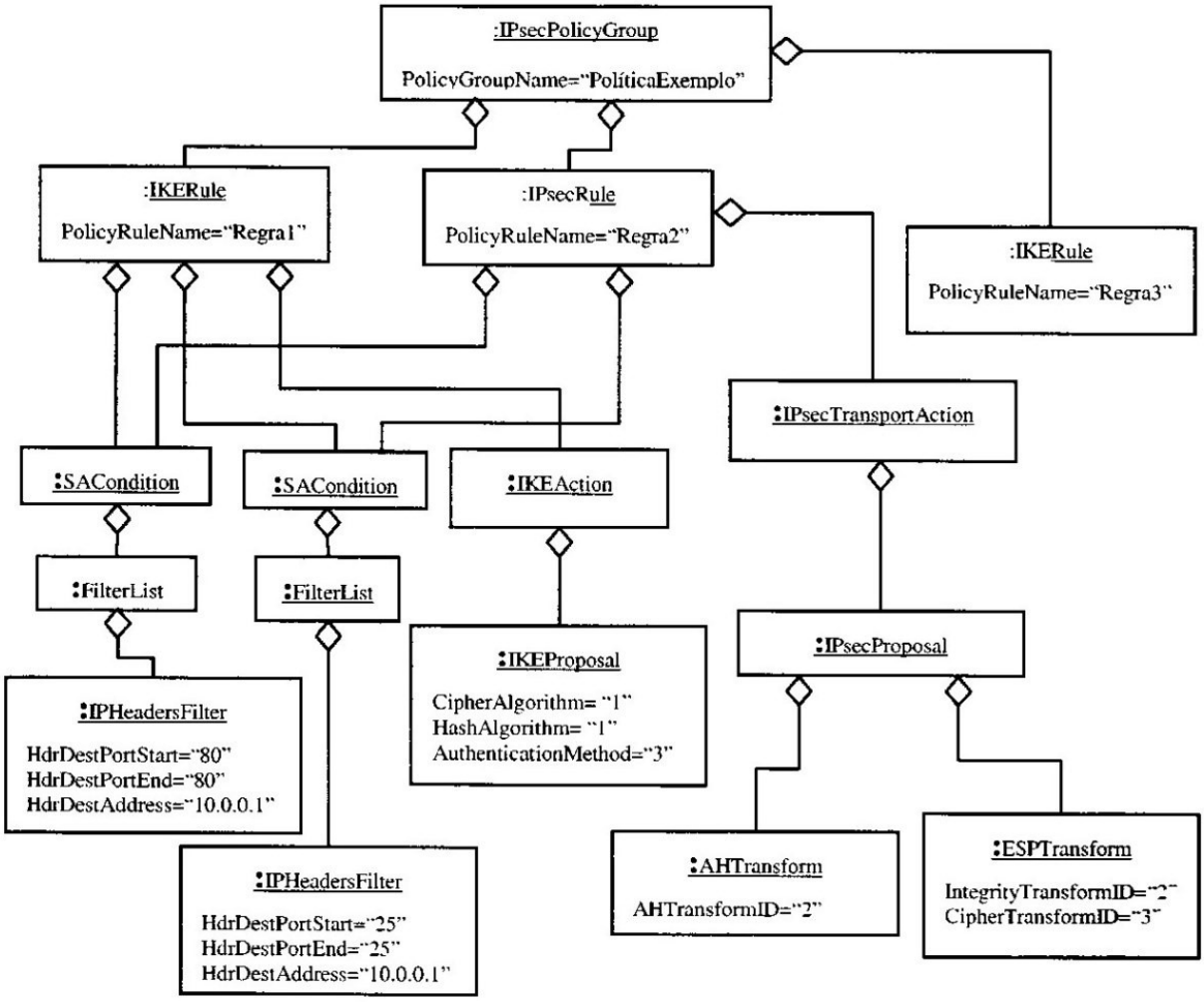

Figura 4.6: Exemplo de Política IPsec

[19] Os números que representam os protocolos são definidos pelo IPsec DOI [PIP98].

4.6. Conclusão do Capítulo

A necessidade de um padrão para representação de políticas motivou o IETF a desenvolver os modelos de políticas apresentado nesse capítulo.

No momento em que essa dissertação foi escrita, o modelo de políticas IETF ainda não estava concluído. Apesar de ainda não estar pronto, esse modelo estava bastante desenvolvido (estava na versão 6) e é bastante rico na representação das características de uma política IPsec, portanto é um bom ponto de partida para o desenvolvimento de quaisquer aplicações ou para a criação de mapeamentos específicos para estruturas de armazenamento.

Por esse motivo, esse modelo foi utilizado como base para o desenvolvimento deste trabalho.

Capítulo 5

Análise do Uso do IPsec com Web Services

5.1. Introdução ao Capítulo

Esse capítulo apresenta alguns itens que devem ser considerados quando o IPsec for utilizado para a criação de canais seguros de comunicação.

O IPsec modifica os pacotes que são transmitidos, portanto é necessário analisar o impacto dessa modificação. Esse impacto pode compreender desde uma diminuição de desempenho até a impossibilidade de envio do pacote, devido à existência de equipamentos de segurança como *firewalls*.

A seção 5.2 trata da configuração de *firewalls* para a utilização do IPsec, apresentando as duas situações possíveis: o tráfego IPsec atravessando o *firewall* (ou seja, o canal foi criado por um computador situado atrás dele) e a criação de um canal IPsec pelo próprio *firewall*. A seção 5.3 descreve alguns cuidados que devem ser tomados relacionados ao desempenho na comunicação, principalmente porque há diferenças entre os tempos de processamento em função dos protocolos utilizados.

O IPsec é bastante flexível com relação aos mecanismos de autenticação. Cada mecanismo tem uma área na qual é mais indicado. A seção 5.4 apresenta uma comparação entre as estratégias de autenticação que podem ser utilizadas com o IPsec, indicando qual estratégia é a mais indica para o uso com os *Web Services*.

A última análise apresentada nesse capítulo está relacionada às configurações estática e dinâmica do IPsec. A seção 5.5 descreve em que casos é mais indicado utilizar cada um dos modos. Por fim, a seção 5.6 apresenta a conclusão do capítulo.

5.2. Configuração de Firewalls

A utilização de mecanismos de proteção é imprescindível. Muitas organizações possuem mecanismos de proteção, que freqüentemente incluem um *firewall*, através do qual sua rede interna é conectada à Internet.

Além disso, qualquer mensagem que seja transmitida pela Internet pode em algum momento ter que atravessar um *firewall*. Esses equipamentos podem permitir ou não a passagem das mensagens, de acordo com a configuração de cada um. Para o IPsec funcionar corretamente, é necessário que as regras do *firewall* permitam a passagem de negociações e mensagens protegidas pelo IPsec.

No destino, onde estarão disponibilizados os *Web Services*, são possíveis os seguintes cenários:

- Canal IPsec criado tendo como destino um *firewall*: É necessário configurar o *firewall* de forma que ele seja capaz de receber pacotes de negociação IKE e criar os canais IPsec de acordo com o especificado na política. Essa configuração deve refletir o que está disponibilizado na política de segurança fornecida com o arquivo WSDL (Figura 5.1 (a)).
- Canal Ipsec tendo como destino um computador protegido por um *firewall*: é necessário configurar o *firewall* de forma a permitir a passagem de todo o tráfego IPsec. Esse tráfego inclui todas as mensagens de negociação do IKE, que tem como origem e destino a porta UDP 500, e os protocolos AH (número de protocolo 51), ESP (número 50) e IPComp (número 108). As regras deverão permitir que esses pacotes se originem ou se destinem ao computador em que estão instalados os *Web Services* (Figura 5.1 (b)). A Tabela 5.1 sumariza as regras de configuração do firewall, supondo a utilização do protocolo HTTP para conexão com os *Web Services*.

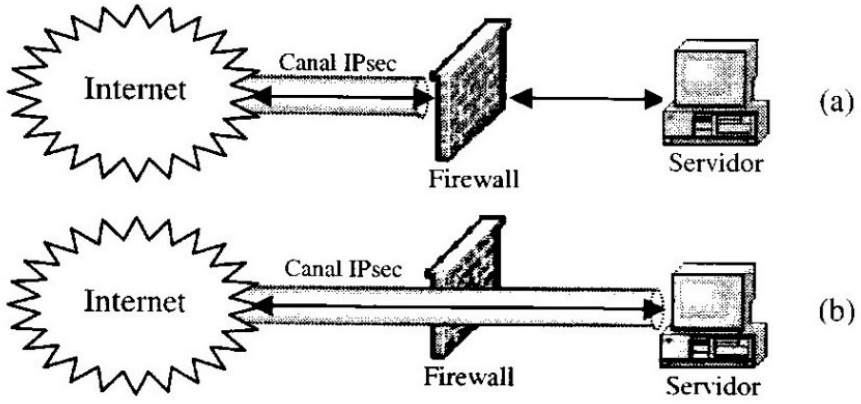

Figura 5.1: Cenários

Tabela 5.1: Regras de Configuração

Direção	**IP Origem**	**IP Destino**	**Protocolo**	**Porta Origem**	**Porta Destino**	**Ação**
Entrada	*	Interno	UDP	500	500	Permitir
Saída	Interno	*	UDP	500	500	Permitir
Entrada	*	Interno	AH	*	*	Permitir
Saída	Interno	*	AH	*	*	Permitir
Entrada	*	Interno	ESP	*	*	Permitir
Saída	Interno	*	ESP	*	*	Permitir
Entrada	*	Interno	IPComp	*	*	Permitir
Saída	Interno	*	IPComp	*	*	Permitir

Na origem, de onde serão realizadas as chamadas para os *Web Services*, são possíveis os seguintes cenários:

- Canal IPsec criado a partir do *firewall*: Da mesma forma que o *firewall* do destino, este *firewall* deve ser configurado de forma que ele crie os canais IPsec. A configuração deve ser feita de forma que ele possa distinguir pacotes com destino ao *Web Service* (ou seja, que devem ser transmitidos através do canal IPsec) e os demais pacotes, que serão enviados para a Internet sem proteção. Se for utilizado o modo dinâmico de configuração do IPsec, a aplicação cliente deve ter acesso e direitos de configuração no *firewall* para aplicar e remover as políticas. No modo estático, o *firewall*

terá todas as configurações. Nesse último caso, se ocorrer uma mudança na política IPsec, o administrador terá que modificar as configurações no *firewall* manualmente para permitir a comunicação.

- Canal IPsec criado a partir de um computador, atrás de um *firewall*: O *firewall* deverá ser configurado para permitir a passagem de todo o tráfego IPsec (pacotes com porta de origem e destino UDP 500 e protocolos de número 50, 51 e 108).

É importante ressaltar que os dados transportados pelo IPsec podem estar cifrados, portanto o *firewall* não tem como determinar o conteúdo dos pacotes de canais que passam por ele. Dessa forma, não é possível criar regras que filtrem por tipo de pacote (por exemplo, permitir somente pacotes que utilizam o protocolo HTTP) ou por algum campo interno do pacote. O *firewall* somente poderá conter regras que filtrem o pacote considerando os dados do protocolo IP e do próprio IPsec.

5.3. Considerações sobre Desempenho

Qualquer método de criptografia empregado causa um retardo na transmissão de informações devido à necessidade de processamento para cifrar e decifrar os dados. Isso também é válido para os métodos de autenticação, pois é necessário calcular o *hash* ao enviar a mensagem e novamente calculá-lo ao recebê-la para verificar a integridade da mensagem.

Em aplicações de tempo real qualquer atraso é significativo e pode comprometer o sistema. Mas para muitas aplicações um atraso não é importante. Aplicações que utilizam a Internet como meio de transporte de informações devem estar preparadas para atrasos.

O IPsec influi no desempenho dos dispositivos que o utilizam. Há algumas razões para isso e podem causar alterações no desempenho [CIS03]:

- O IPsec expande o tamanho do pacote, o que pode requerer fragmentação e a remontagem de pacotes;
- Os pacotes cifrados provavelmente serão autenticados, o que implica na execução de duas operações criptográficas para cada pacote;
- Os mecanismos de autenticação dos dispositivos IPsec são lentos.

Além disso, o algoritmo de troca de chaves do Diffie-Hellman utilizado pelo IKE é uma exponenciação de números muito grandes (entre 768 e 1024 bytes), cujo processamento é complexo e o tempo gasto para isso pode ser significativo dependendo do

equipamento utilizado (em um roteador Cisco 2500 pode levar até 4 segundos [CIS03]). Outro item que afeta o desempenho é o algoritmo RSA (utilizado para autenticação dos dispositivos), que é dependente do tamanho do número primo escolhido para a geração do par de chaves RSA.

Mesmo os pacotes que não são cifrados terão um atraso quando passarem por um dispositivo com o IPsec ativo, pois todos os pacotes são testados pelas regras para determinar quais devem ser processados pelo IPsec.

[MCG00] procurou avaliar o tempo gasto para preparar os pacotes IP (cifrá-los e calcular o *hash*) do envio, para transmití-los através do meio físico (rede) e para interpretar os pacotes após a recepção (decifrá-los e recalcular o *hash* para verificação de integridade).

Nessas avaliações, concluiu-se que para baixas velocidades de transmissão o tempo gasto na preparação e na interpretação não afeta significativamente o tempo total (considerando uma conexão discada de 56 kbps). Para conexões de até 1.54 Mbps a degradação fica abaixo de 10%. À medida que a velocidade vai aumentando, a preparação e a interpretação começam a ser mais significativos, podendo chegar até 98% dependendo da combinação dos protocolos de criptografia e de autenticação.

Considerando a utilização da Internet como meio de transmissão, a utilização de criptografia e autenticação não deverá causar grande impacto, já que as velocidades de transmissão não são grandes.

Caso exista alguma aplicação em que o tempo de transmissão seja uma característica importante, pode-se ainda fazer uma escolha mais apurada dos protocolos de criptografia e autenticação.

De acordo com as avaliações de [ELK02] e [MCG00], o algoritmo de autenticação MD5 é mais rápido que o algoritmo SHA-1, e o algoritmo de criptografia RC5 é mais rápido que o algoritmo 3DES. Apesar dos algoritmos MD5 e RC5 serem menos seguros, nem sempre é necessária uma segurança muito forte. Além disso, o tempo gasto com a criptografia é muito maior que o tempo gasto com a autenticação dos pacotes. Portanto, quando o desempenho for importante, somente deve ser utilizada a criptografia se for realmente imprescindível.

Existe ainda a possibilidade de utilizar a compressão, mas o tempo computacional gasto para a sua realização limita a sua utilização. Geralmente a aplicação da compressão

melhora o desempenho para redes com velocidade de até 10Mbps. Para velocidades superiores o desempenho somente melhora em algumas combinações de algoritmos.

Para minimizar o impacto do processamento IPsec, [CIS03] sugere utilizar o grupo 1 para as trocas de chave do Diffie-Hellman, utilizar o MD5 para algoritmo de autenticação, RC5 para criptografia e utilizar tempos de vida maiores. Com essa configuração, acaba-se obtendo uma proteção mais fraca. É necessário consultar a política de segurança da organização para determinar e verificar quais algoritmos e valores podem ser utilizados.

5.4. Análise das Estratégias de Autenticação do IPsec

O IKE é bastante flexível e suporta mais de um método de autenticação como parte da negociação da fase 1. Os dois dispositivos IPsec precisam negociar e concordar com um protocolo de autenticação comum.

O IKE suporta 5 tipos de autenticação, e para cada tipo é possível utilizar 2 modos: o principal ou o agressivo. Os possíveis métodos de autenticação são [HAR98]:

- Chaves pré-compartilhadas: A mesma chave é configurada para pares IPsec (dois dispositivos que estabelecerão uma sessão). O IKE autentica o outro dispositivo enviando uma mensagem com o *hash* de dados que incluem a chave pré-compartilhada. Se o dispositivo que receber essa mensagem for capaz de criar independentemente o mesmo valor *hash* utilizando a chave pré-compartilhada, então ambos os dispositivos compartilham a mesma chave, portanto autenticando um ao outro. Esse método de autenticação é mais simples de configurar, mas não é muito escalável, pois cada dispositivo rodando IPsec deve ser configurado com a chave pré-compartilhada de todos os dispositivos com os quais serão criadas sessões.
- Assinaturas digitais (RSA ou DSS): Neste método cada dispositivo deverá assinar um conjunto de dados (utilizando a sua chave privada) e enviar para o outro. A autenticação ocorre quando esse conjunto de dados for decifrado utilizando a chave pública do outro dispositivo. As assinaturas digitais usam autoridades certificadoras (CA – *Certificate Authority*) para gerar um certificado digital para cada computador que precisa ser autenticado. O método de autenticação com certificado digital funciona de

forma similar ao método de chaves pré-compartilhadas, mas provê uma segurança muito maior.

- Criptografia de chave pública (RSA): Esse método requer que cada dispositivo IPsec gere um número pseudo-aleatório (chamado *nonce*) e o cifre utilizando a chave pública RSA do outro dispositivo. A autenticação ocorre quando cada parte decifra o número gerado pelo outro dispositivo utilizando a chave privada e o utiliza para criar um ***hash***. Esse método não provê não-repudiação, ou seja, um dos dispositivos pode negar ter participado da negociação. Isso acontece porque qualquer dispositivo pode reconstruir todas as mensagens.
- Criptografia de chave pública (RSA) revisada: Esse método foi incorporado para reduzir o número de operações com chaves assimétricas (cifragem/decifragem com criptografia de chave pública), pois esse tipo de operação demanda muito processamento. O modo original requer que cada dispositivo execute duas operações com chaves públicas (duas cifragens e duas decifragens), e este modo requer apenas um.
- Kerberos [KOH93]: O dispositivo que iniciou a negociação IPsec obtém um *ticket* Kerberos[20] (solicitado ao KDC[21]) e envia ao outro dispositivo. O outro dispositivo armazena esse *ticket*, obtém outro *ticket* e o envia para o primeiro dispositivo. Após isso, cada dispositivo cria um *hash*, que é assinado com base no *ticket*, e enviado para o outro lado. A autenticação ocorre quando cada lado valida a assinatura do *hash* recebido.

A Tabela 5.2 apresenta um comparativo entre os métodos e apresenta as vantagens e desvantagens de cada um [MUR99].

[20] O *ticket* Kerberos é um registro obtido do KDC utilizado pelo cliente para se autenticar no servidor. Contém a identidade do cliente, uma chave de sessão e outras informações, todas protegidas pela chave secreta do servidor.
[21] KDC – Key Distribution Center – um serviço de rede que provê *tickets* e as chaves de sessão temporárias.

Tabela 5.2: Comparação entre Métodos de Autenticação

Método de autenticação	**Como a autenticação é realizada**	**Vantagens**	**Desvantagens**
Chaves pré-compartilhadas	Criando *hash*´s das informações trocadas	Simples	A chave deve ser distribuída por outro meio antes das negociações iniciarem Somente o endereço IP pode ser utilizado como Identidade
Assinaturas digitais (RSA ou DSS)	Assinando *hash*´s criados com base nas informações trocadas (assina utilizando a própria chave privada)	Pode usar Identidades diferentes do endereço IP Não é necessário ter o certificado do outro dispositivo IPsec antes de iniciar as negociações	Requer operações com certificados
Criptografia de chave pública (RSA)	Cifrando *hash*´s sobre valores pseudo-aleatórios com a chave pública do outro dispositivo IPsec	Melhor segurança por adicionar operações de chave pública às trocas Diffie-Hellman Permite proteção do ID no modo agressivo	As chaves públicas (certificados) precisam estar disponíveis antes de iniciar as negociações Grande processamento devido às operações com chaves públicas

Tabela 5.2: Comparação entre Métodos de Autenticação (cont.)

Método de autenticação	**Como a autenticação é realizada**	**Vantagens**	**Desvantagens**
Criptografia de chave pública (RSA) revisado	Idem ao item anterior	Idem ao item anterior Menor número de operações com chave pública	As chaves públicas (certificados) precisam estar disponíveis antes de iniciar as negociações
Kerberos	Cliente e servidor obtêm *tickets kerberos* e os utilizam para autenticar-se mutuamente	Simples de ser configurado em um único domínio	Não é indicado para aplicações distribuídas na Internet, pois necessita de um servidor central ou de servidores interligados Pode ter problemas de interoperabilidade entre o MS[22] Windows 2000 e outros sistemas operacionais (o Kerberos da MS não segue o padrão)

O método de chaves pré-compartilhadas não deve ser escolhido para aplicações *Web Services*. Não é seguro utilizar esse método para esse tipo de aplicação, na qual a chave é disponibilizada junto com a descrição do serviço. O objetivo da autenticação é provar que quem está enviando as informações é realmente quem ele diz ser, mas como qualquer um pode ter obtido essa chave não há como garantir a autenticidade.

[22] Microsoft

O método de autenticação mais indicado para ser utilizado com *Web Services* é o de assinaturas digitais. Alguns autores recomendam inclusive que o método de criptografia de chave pública seja retirado da especificação do IPsec [HAR02] [KAU01] [PER01].

5.5. Criação de Regras Dinâmicas

As implementações do IPsec permitem que a configuração seja realizada de duas formas: estática ou dinâmica.

As regras estáticas criam ou modificam a base de políticas do IPsec. Esse tipo é utilizado para regras que serão utilizadas várias e várias vezes, até que alguma modificação seja realizada ou enquanto a base de políticas existir. Isso significa que a regra não será perdida (ou desativada) mesmo reiniciando o dispositivo ou parando e reiniciando o serviço IPsec. Esta é a configuração que deverá ser utilizada em servidores de aplicação.

As regras dinâmicas são regras que existem somente no serviço IPsec (não estão armazenadas na base de políticas) e que estão ativas somente enquanto o serviço estiver ativo. Se o serviço for reiniciado ou se o computador for desligado essas regras serão perdidas. Essas regras normalmente sobrepõem as regras estáticas (esta característica pode variar de acordo com a implementação do sistema operacional). Esse tipo de conexão pode ser utilizado para a criação de conexões que duram um curto período de tempo, por exemplo, clientes que desejam realizar algumas chamadas ao servidor de aplicação e que depois disso desejam desfazer a configuração por motivos de segurança.

Para aplicações distribuídas na Internet, em que o canal é criado apenas por alguns instantes, é recomendável a utilização de regras dinâmicas, pois permitem a conexão somente durante o período necessário para a realização das requisições/tarefas e não afetam a base de políticas.

5.6. Conclusão do Capítulo

O presente capítulo apresentou uma análise da utilização do IPsec para prover segurança de *Web Services*.

A seção 5.2 apresentou as alterações necessárias nos *firewalls* para permitir que a negociação do canal IPsec possa ocorrer e os pacotes protegidos sejam capazes de chegar ao seu destino.

O impacto sobre o desempenho causado pelo IPsec foi apresentado na seção 5.3. Em aplicações em que um pequeno atraso (na ordem de segundos) não é significante, não há a necessidade de se preocupar com a degradação do desempenho. Mas em aplicações de tempo real a degradação pode ser significativa e portanto é necessário escolher uma combinação de protocolos e algoritmos que mantenha a segurança sem diminuir tanto o desempenho.

A seção 5.4 fez um comparativo entre os métodos de autenticação do IPsec, indicando as vantagens e desvantagens de cada um, mostrando qual o método mais indicado para ser utilizado com *Web Services*.

Por fim, a seção 5.5 apresentou os modos de configuração do IPsec, descrevendo em que caso utilizar cada modo.

Capítulo 6

Proposta de Mecanismo de Segurança para Web Services

6.1. Introdução ao Capítulo

Este capítulo apresenta um mecanismo para trabalhar com políticas de segurança IPsec. Esse mecanismo é composto por um esquema XML para representação de políticas IPsec, uma API[23] para carregar e aplicar as políticas e de um documento para representação de requisitos mínimos de políticas do cliente.

A API permite a utilização das políticas IPsec pelos clientes sem implicar em grandes modificações em seus sistemas legados. A aplicação, através dos métodos disponibilizados, pode obter, aplicar e remover as políticas sem necessitar tratar diretamente o documento de políticas. A própria API, utilizando um arquivo de requisitos mínimos fornecido pela aplicação, extrai as regras mais adequadas para realizar a proteção.

O arquivo de requisitos mínimos conterá as políticas permitidas pelo cliente para criação dos canais IPsec. Esse arquivo pode conter diversas políticas, que deverão aparecer em ordem de prioridade. O *Web Service* pode fornecer um conjunto grande de políticas, mas a API irá extrair apenas uma entre as que satisfizerem os requisitos mínimos (baseado nas prioridades das políticas). Isso evita que seja necessário negociação no momento da criação do canal IPsec além de assegurar que a melhor política será sempre a escolhida.

A seção 6.2 apresenta uma descrição do mecanismo proposto. A seção 6.3 apresenta o mapeamento de cada classe do modelo IETF para o esquema XML. A seção 6.4 apresenta a proposta de extensão do WSDL para suportar a representação de políticas IPsec.

[23] API – Application Program Interface

A seção 6.5 apresenta um exemplo de política de segurança utilizando o esquema proposto. A seção 6.6 apresenta o documento de requisitos mínimos de segurança do cliente. A seção 6.7 apresenta a API para desenvolvimento de aplicações. Por fim, a seção 6.8 conclui o capítulo.

6.2. Mecanismo Proposto

A Figura 6.1 apresenta a interação entre a API e as demais partes.

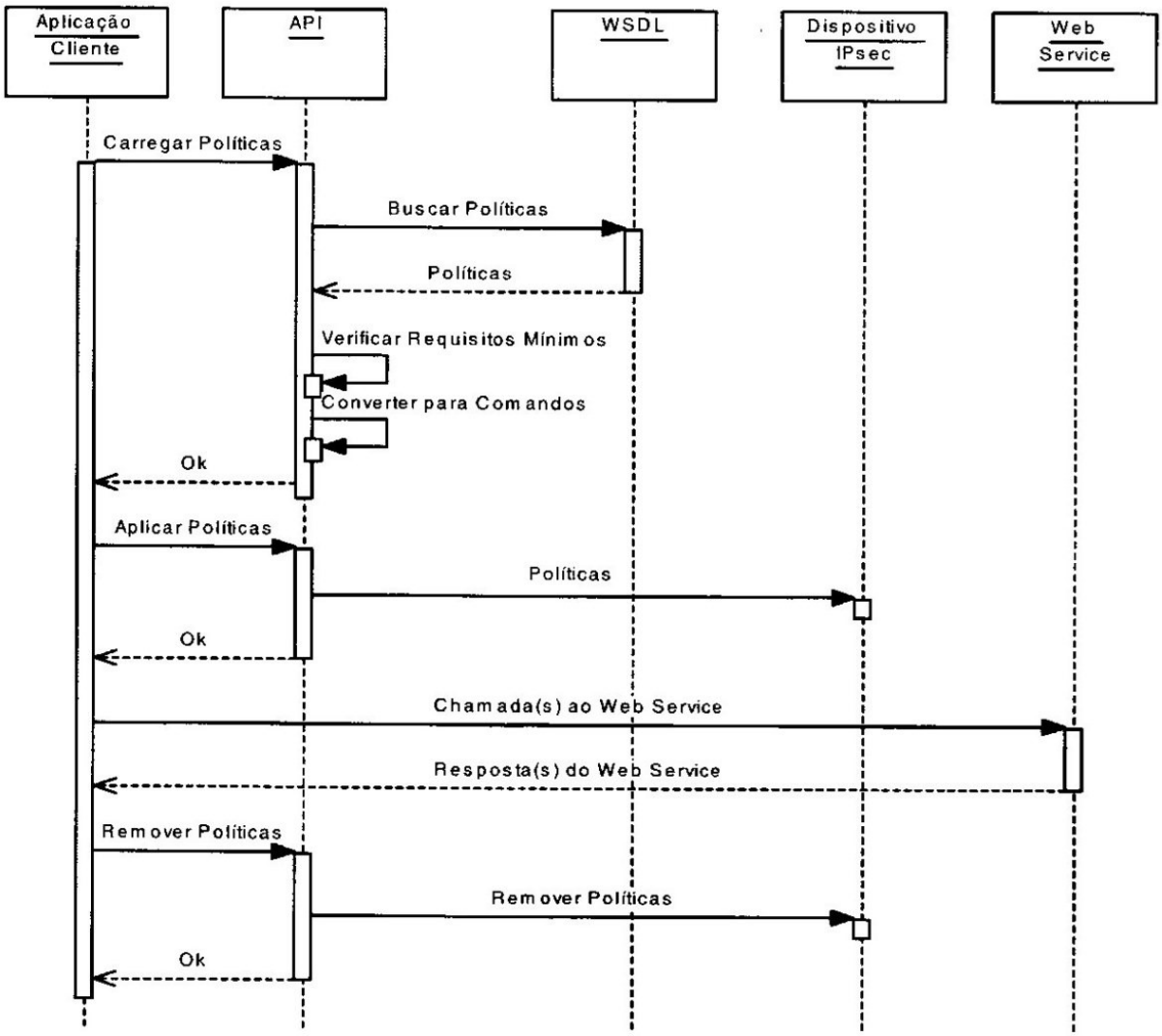

Figura 6.1: Diagrama de Seqüência

A aplicação cliente somente precisa interagir com a API e com o próprio *Web Service*. A API é responsável por obter as políticas de segurança e configurar adequadamente o dispositivo IPsec.

O processo é iniciado com a aplicação solicitando à API para carregar o documento WSDL com as políticas de segurança (indicado como parâmetro). Após a API obter esse documento, dois processos são executados: verificação e obtenção dos requisitos mínimos e conversão da política para comandos de configuração do dispositivo IPsec. A aplicação poderá então chamar o método de aplicação das políticas, que executará os comandos de configuração. Nesse momento a aplicação poderá realizar as chamadas ao *Web Service* normalmente. O próprio dispositivo se encarrega de criar o canal IPsec.

Quando a aplicação terminar de realizar as chamadas ao *Web Service*, ela deverá chamar o método de remoção das políticas IPsec. Dessa forma, o dispositivo volta a ter a configuração original.

6.3. Esquema XML para Representação de Políticas IPsec

Esta seção apresenta uma estrutura de documento XML para representar as políticas de segurança de IPsec. O documento criado com base nessa estrutura será disponibilizado juntamente com o documento WSDL que descreve o serviço, para que os desenvolvedores de aplicações clientes possam configurar as regras IPsec de seu sistema para troca segura de informações com os *Web Services*.

A estrutura de documento XML proposta é baseada no modelo de políticas IPsec apresentado no capítulo 4. Esse modelo é composto por dois tipos de classes: classes estruturais, que representam as informações sobre as políticas, e classes associativas, que além de prover informações complementares sobre as políticas indicam como as instâncias das classes estruturais se relacionam umas com as outras.

Dois tipos de mapeamento do modelo para a estrutura proposta são necessários:

- Mapeamento de classes estruturais: o mapeamento é basicamente um-para-um, para as classes mais específicas. No mapeamento, os elementos representando as classes específicas recebem todos os atributos das classes hierarquicamente superiores (classes genéricas não são representadas).
- Mapeamento de classes associativas: Há várias alternativas, mas nesse trabalho foram utilizadas duas: definir elementos para as classes estruturais e mapear os atributos das classes associativas para esses elementos (o elemento que recebe os atributos é definido de acordo com a semântica de

cada associação e seus atributos) ou definir um elemento para representar classe associativa.

Assim como no modelo definido pelo IETF, os atributos que indicam tipos e protocolos armazenam números, e não os nomes. A relação entre esses números e os tipos/protocolos é definida no IPsec DOI [PIP98].

6.3.1. Definição do Elemento Raiz

O elemento raiz, chamado de SecurityPolicy (Figura 6.2), é utilizado para agrupar um conjunto de políticas. Este elemento é definido como uma seqüência de um ou mais elementos IPsecPolicyGroup, definidos pelo tipo IPsecPolicyGroupType (Figura 6.3).

O elemento SecurityPolicy possui os seguintes atributos:

- SecurityPolicyName: Nome da política de segurança.
- PolicyDecisionStrategy: Método de avaliação utilizado para as políticas contidas nessa política de segurança. Os valores possíveis são: 1 – somente a primeira política encontrada cujas condições foram satisfeitas deve ser considerada, e 2 – todas as políticas devem ser avaliadas e todas aquelas cujas condições forem satisfeitas devem ser executadas. Se for omitido, será assumido o valor 1 (primeira política satisfeita).

```
<xsd:element name="SecurityPolicy">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="IPsecPolicyGroup" type="IPsecPolicyGroupType"
            maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="SecurityPolicyName" type="xsd:string"/>
      <xsd:attribute name="PolicyDecisionStrategy"
         type="xsd:nonNegativeInteger"/>
   </xsd:complexType>
</xsd:element>
```

Figura 6.2: Elemento Raiz para Definição de Segurança

6.3.2. Políticas

As políticas são representadas pela classe IPsecPolicyGroup no modelo IETF. Para representá-las em XML, foi criado o elemento IPsecPolicyGroup, que no esquema XML é definido pelo IPsecPolicyGroupType.

As políticas contêm um conjunto de regras, que podem ser da negociação da fase 1 (classe IKERule) ou fase 2 (IPsecRule). Essas duas classes são representadas pelos

elementos IKERule e IPsecRule respectivamente, e representadas no esquema por IKERuleType (Figura 6.5) e IPsecRuleType (Figura 6.6).

O modelo IETF permite ainda a representação de regras aninhadas. Isso é possível através da classe associativa PolicySetComponent, da qual a classe IPsecPolicyGroup é derivada. Para permitir isso, o esquema prevê que um IPsecPolicyGroup contenha elementos do mesmo tipo, representando dessa forma políticas aninhadas. A classe PolicySetComponent possui um atributo Priority, que é utilizada para indicar a prioridade das políticas. Esse atributo foi incluído no elemento IPsecPolicyGroup.

A Figura 6.3 apresenta o esquema XML do elemento IPsecPolicyGroup.

```
<xsd:complexType name="IPsecPolicyGroupType">
   <xsd:sequence>
      <xsd:element name="IKERule" type="IKERuleType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="IPsecRule" type="IPsecRuleType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="IPsecPolicyGroup" type="IPsecPolicyGroupType"
         minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyGroupName" type="xsd:string"/>
   <xsd:attribute name="PolicyDecisionStrategy"
      type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="PolicyRoles" type="xsd:string"/>
   <xsd:attribute name="Priority" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.3: Esquema XML dos Elementos de Políticas

Os atributos PolicyDecisionStrategy e Priority determinam a estratégia de processamento das políticas.

A Figura 6.4 mostra um exemplo de hierarquia de política e regras para construir uma estratégia. O PolicyGroup de nome P1 contém duas sub-políticas e uma regra. Esse PolicyGroup possui o atributo PolicyDecisionStrategy igual a 1, ou seja, a primeira sub-política cujas regras sejam satisfeitas terá suas ações executadas e o processamento será encerrado.

O IPsecPolicyGroup P1Pa, que é a primeira sub-política de P1, tem duas regras. Como o atributo PolicyDecisionStrategy de P1Pa é igual a 2, todas as regras têm que ter suas condições avaliadas. As ações das regras que fazem parte dessa política cujas condições forem satisfeitas são executadas.

Caso a política P1Pa não seja satisfeita, então deve ser avaliada a regra P1Rb. Novamente, se essa regra for satisfeita não é necessário continuar a avaliação das demais

políticas e regras. Caso P1Rb não seja satisfeita, deve-se então avaliar a próxima sub-política, no caso P1Pc, que possui duas regras. Pelo fato do atributo PolicyDecisionStrategy de P1Pc ser igual a 1, a primeira regra que for satisfeita será executada e o processamento termina.

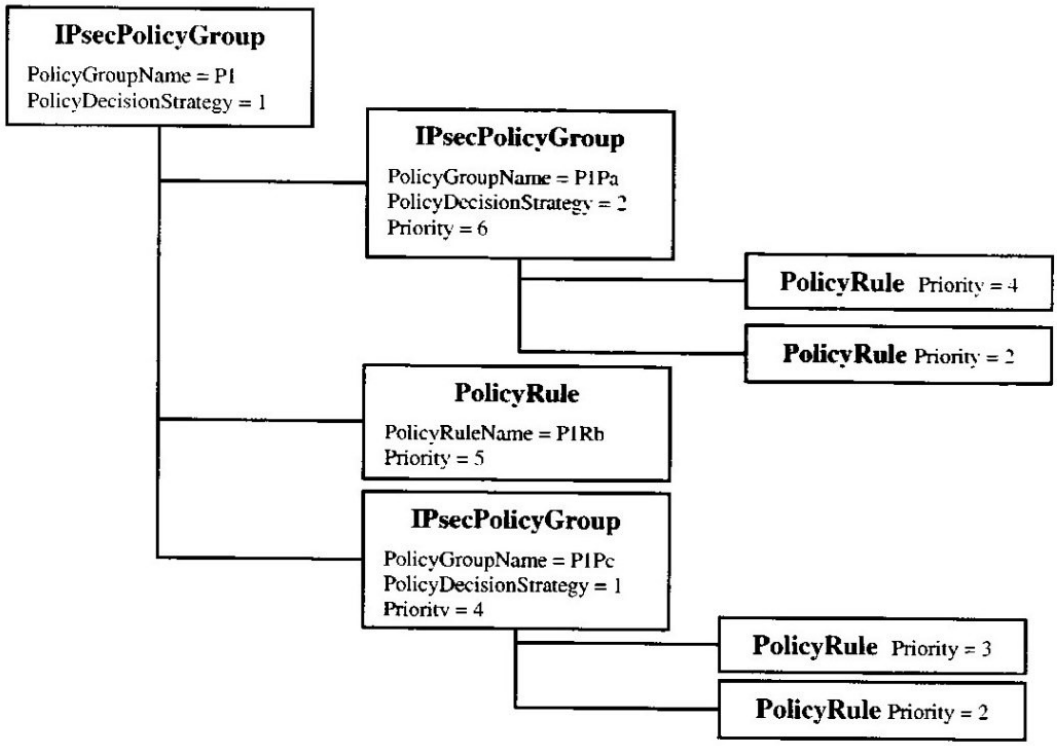

Figura 6.4: Exemplo de Estratégia de Decisão

6.3.3. Regras

No modelo IETF, as regras são representadas pelas classes SARule, IKERule e IPsecRule. A primeira é uma generalização das outras duas, e possui os atributos que são comuns a elas. Para representar as regras em XML, foram definidos dois elementos: IKERule (Figura 6.5) e IPsecRule (Figura 6.6), sendo que os dois contêm todos os atributos das classes de mesmo nome mais os atributos herdados da classe SARule.

As regras contêm um conjunto de condições e ações. Apesar das ações serem genericamente associadas às regras através da classe SARule, o modelo IETF define que instâncias da classe IKERule devem ser associadas somente a instâncias das classes de ações IKE (IKEAction, IKERejectAction) e que instâncias da classe IPsecRule devem ser associadas somente a instâncias das classes de ações IPsec (IPsecTransportAction, IPsecTunnelAction, IPsecBypassAction, IPsecDiscardAction).

Devido a essa restrição, o esquema XML foi definido da seguinte forma: um elemento IKERule que contém como sub-elementos IKEAction, IKERejectAction e

PreconfiguredSAAction, e um elemento IPsecRule que contém como sub-elementos IPsecTransportAction, IPsecTunnelAction, IPsecBypassAction, IPsecDiscardAction e PreconfiguredSAAction.

Os dois tipos de regras podem conter os mesmos tipos de condições e filtros, portanto somente um tipo de elemento de condição foi definido: SAConditionType.

Da mesma forma que as políticas, é possível construir regras aninhadas (pelo fato da classe SARule ser derivada da classe PolicySet). Portanto, os elementos de regras podem conter como sub-elementos outras regras (respeitando o tipo de regra, isto é, se a regra é do tipo IKE então as sub-regras também devem ser desse tipo). O atributo Priority da classe PolicySetComponent foi incluído nos elementos IKERule e IPsecRule.

```
<xsd:complexType name="IKERuleType">
   <xsd:sequence>
      <xsd:element name="SACondition" type="SAConditionType"
         maxOccurs="unbounded"/>
      <xsd:element name="PolicyTimePeriodCondition"
         type="PolicyTimePeriodConditionType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="IKEAction" type="IKEActionType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="IKERejectAction" type="IKERejectActionType"
         minOccurs="0"/>
      <xsd:element name="PreconfiguredSAAction"
         type="PreconfiguredSAActionType" minOccurs="0"/>
      <xsd:element name="CompoundIKEAction" type="CompoundIKEActionType"
         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="IKERule" type="IKERuleType" minOccurs="0"
         maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyRuleName" type="xsd:string"/>
   <xsd:attribute name="Enable" type="xsd:boolean"/>
   <xsd:attribute name="ConditionListType" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="RuleUsage" type="xsd:string"/>
   <xsd:attribute name="Mandatory" type="xsd:boolean"/>
   <xsd:attribute name="SequencedActions" type="xsd:string"/>
   <xsd:attribute name="ExecutionStrategy" type="xsd:string"/>
   <xsd:attribute name="PolicyRoles" type="xsd:string"/>
   <xsd:attribute name="PolicyDecisionStrategy"
      type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="LimitNegotiation" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="IdentityContexts" type="xsd:string"/>
   <xsd:attribute name="Priority" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.5: Esquema XML do Elemento IKERule

```
<xsd:complexType name="IPsecRuleType">
   <xsd:sequence>
      <xsd:element name="SACondition" type="SAConditionType"
         maxOccurs="unbounded"/>
      <xsd:element name="PolicyTimePeriodCondition"
         type="PolicyTimePeriodConditionType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="IPsecTransportAction"
         type="IPsecTransportActionType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="IPsecTunnelAction" type="IPsecTunnelActionType"
         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="IPsecBypassAction" type="IPsecBypassActionType"
         minOccurs="0"/>
      <xsd:element name="IPsecDiscardAction" type="IPsecDiscardActionType"
         minOccurs="0"/>
      <xsd:element name="PreconfiguredSAAction"
         type="PreconfiguredSAActionType" minOccurs="0"/>
      <xsd:element name="CompoundIPsecAction" type="CompoundIPsecActionType"
         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="IPsecRule" type="IPsecRuleType" minOccurs="0"
         maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyRuleName" type="xsd:string"/>
   <xsd:attribute name="Enable" type="xsd:boolean"/>
   <xsd:attribute name="ConditionListType" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="RuleUsage" type="xsd:string"/>
   <xsd:attribute name="Mandatory" type="xsd:boolean"/>
   <xsd:attribute name="SequencedActions" type="xsd:string"/>
   <xsd:attribute name="ExecutionStrategy" type="xsd:string"/>
   <xsd:attribute name="PolicyRoles" type="xsd:string"/>
   <xsd:attribute name="PolicyDecisionStrategy"
      type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="LimitNegotiation" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="Priority" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.6: Esquema XML do Elemento IPsecRule

O modelo permite que um elemento de regra contenha sub-regras. A relação entre as regras e suas sub-regras é a seguinte:

- As condições da regra devem ser satisfeitas antes de realizar a avaliação das sub-regras. É equivalente a aplicar o operador lógico AND entre as condições da regra e as condições de cada sub-regra.
- Se a condição da regra for satisfeita, então o conjunto de sub-regras deve ser avaliado de acordo com a estratégia de decisão e as prioridades.
- Se a condição da regra for satisfeita, então as ações da regra devem ser executadas antes da execução das ações das sub-regras.

6.3.4. Filtros e Condições

As regras contêm um conjunto de condições (através da agregação SAConditionInRule), que são representadas pelos elementos SACondition (definido pelo tipo

SAConditionType, Figura 6.7) e PolicyTimePeriodCondition (definido pelo tipo PolicyTimePeriodConditionType, Figura 6.8).

A classe de agregação SAConditionInRule possui 2 atributos, GroupNumber e ConditionNegated, que foram mapeadas para a classe SACondition e PolicyTimePeriodCondition.

```
<xsd:complexType name="SAConditionType">
   <xsd:sequence>
      <xsd:element name="FilterList">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="IPSOFilterEntry" minOccurs="0"
                  maxOccurs="unbounded"> ...     </xsd:element>
               <xsd:element name="PeerIDPayloadFilterEntry" minOccurs="0"
                  maxOccurs="unbounded"> ...     </xsd:element>
               <xsd:element name="CredentialFilterEntry" minOccurs="0"
                  maxOccurs="unbounded"> ...     </xsd:element>
               <xsd:element name="IPHeadersFilter" minOccurs="0"
                  maxOccurs="unbounded"> ...     </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="Direction" type="xsd:nonNegativeInteger"/>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="AcceptCredentialsFrom" minOccurs="0">
         ...
      </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="PolicyConditionName" type="xsd:string"/>
   <xsd:attribute name="GroupNumber" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="ConditionNegated" type="xsd:boolean"/>
</xsd:complexType>
```

Figura 6.7: Esquema XML do Elemento SACondition

O elemento PolicyTimePeriodCondition, que representa a classe de mesmo nome, tem o objetivo de especificar o período de tempo em que as regras são válidas.

```
<xsd:complexType name="PolicyTimePeriodConditionType">
   <xsd:attribute name="PolicyConditionName" type="xsd:string"/>
   <xsd:attribute name="ConditionNegated" type="xsd:boolean"/>
   <xsd:attribute name="TimePeriod" type="xsd:string"/>
   <xsd:attribute name="MonthOfYearMask" type="xsd:string"/>
   <xsd:attribute name="DayOfMonthMask" type="xsd:string"/>
   <xsd:attribute name="DayOfWeekMask" type="xsd:string"/>
   <xsd:attribute name="TimeOfDayMask" type="xsd:string"/>
   <xsd:attribute name="LocalOrUtcTime" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.8: Esquema XML do Elemento PolicyTimePeriodCondition

Cada elemento SACondition possui uma lista de filtros (representadas pelo elemento FilterList), sendo que a lista de filtros é composta por um ou mais filtros

representados pelos elementos IPSOFilterEntry (Figura 6.9), PeerIDPayloadFilterEntry (Figura 6.10), CredentialFilterEntry (Figura 6.11) e IPHeadersFilter (Figura 6.12). Vários filtros dentro de uma lista de filtros são avaliados aplicando-se o operador lógico AND.

O elemento IPSOFilterEntry (Figura 6.9) é utilizado para filtrar o tráfego baseado nos valores do cabeçalho IP *Security Options*. Os atributos Name e IsNegated foram herdados de FilterEntry. Os atributos MatchConditionType e MatchConditionValue indicam o campo do cabeçalho que será testado e o valor que este campo deve conter.

```
<xsd:element name="IPSOFilterEntry" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:attribute name="Name" type="xsd:string"/>
      <xsd:attribute name="IsNegated" type="xsd:boolean"/>
      <xsd:attribute name="MatchConditionType" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="MatchConditionValue" type="xsd:nonNegativeInteger"/>
   </xsd:complexType>
</xsd:element>
```

Figura 6.9: Esquema XML do Elemento IPSOFilterEntry

O elemento PeerIDPayloadFilterEntry (Figura 6.10) é utilizado para filtrar o tráfego baseado no ID *payload* do protocolo IKE. Também possui os atributos Name e IsNegated herdados de FilterEntry. Os atributos MatchConditionType especificam o tipo de identidade IKE a ser filtrado e MatchConditionValue o valor que a identidade deve conter.

```
<xsd:element name="PeerIDPayloadFilterEntry" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:attribute name="Name" type="xsd:string"/>
      <xsd:attribute name="IsNegated" type="xsd:boolean"/>
      <xsd:attribute name="MatchConditionType" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="MatchConditionValue" type="xsd:string"/>
   </xsd:complexType>
</xsd:element>
```

Figura 6.10: Esquema XML do Elemento PeerIdPayloadFilterEntry

O elemento CredentialFilterEntry (Figura 6.11) é utilizado para filtrar credenciais de dispositivos IKE. O tipo da credencial é especificado pelo atributo CredentialType, indicando certificados X.509 ou *tickets* Kerberos. O campo do certificado a ser testado e o valor que este campo deve conter são especificados pelos atributos MatchFieldName e MatchFieldValue.

```
<xsd:element name="CredentialFilterEntry" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:attribute name="Name" type="xsd:string"/>
      <xsd:attribute name="IsNegated" type="xsd:boolean"/>
      <xsd:attribute name="MatchFieldName" type="xsd:string"/>
      <xsd:attribute name="MatchFieldValue" type="xsd:string"/>
      <xsd:attribute name="CredentialType"
         type="xsd:nonNegativeInteger"/>
   </xsd:complexType>
</xsd:element>
```

Figura 6.11: Esquema XML do Elemento CredentialFilterEntry

O elemento IPHeadersFilter (Figura 6.12) é utilizado para filtrar pacotes IP baseado em endereços, portas, protocolos e outras propriedades do pacote IP. Por exemplo, os atributos HdrSrcAddress e HdrScrcAddressEndOfRange são utilizados para especificar a faixa de endereços de origem, enquanto que os atributos HdrDestAddress e HdrDestAddressEndOfRange especificam a faixa de endereços de destino. O atributo HdrFlowLabel foi mapeado como um sub-elemento pois ele é uma lista de números inteiros, portanto, a representação na forma de um atributo de um elemento XML não é adequada. Os demais atributos são detalhados no Anexo A.

```
<xsd:element name="IPHeadersFilter" minOccurs="0"
   maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="HdrFlowLabel" minOccurs="0">
            <xsd:simpleType>
               <xsd:list itemType="xsd:nonNegativeInteger"/>
            </xsd:simpleType>
         </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="Name" type="xsd:string"/>
      <xsd:attribute name="IsNegated" type="xsd:boolean"/>
      <xsd:attribute name="HdrIpVersion" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="HdrSrcAddress" type="xsd:string"/>
      <xsd:attribute name="HdrSrcAddressEndOfRange" type="xsd:string"/>
      <xsd:attribute name="HdrSrcMask" type="xsd:string"/>
      <xsd:attribute name="HdrDestAddress" type="xsd:string"/>
      <xsd:attribute name="HdrDestAddressEndOfRange" type="xsd:string"/>
      <xsd:attribute name="HdrDestMask" type="xsd:string"/>
      <xsd:attribute name="HdrProtocolID" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="HdrSrcPortStart" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="HdrSrcPortEnd" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="HdrDestPortStart" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="HdrDestPortEnd" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="HdrDSCP" type="xsd:nonNegativeInteger"/>
   </xsd:complexType>
</xsd:element>
```

Figura 6.12: Esquema XML do Elemento IPHeadersFilter

O modelo IETF divide as condições em dois grupos: específicas a uma regra (ou seja, associadas a uma única regra) ou reutilizáveis (associadas a diversas regras). Não há diferenças entre a estrutura e o conteúdo entre os dois grupos. A diferença está no tratamento dessas condições [MOO01]. O mapeamento para o esquema XML foi realizado da seguinte forma:

- Condições específicas a uma regra: definem-se normalmente todos os atributos e sub-elementos da condição.
- Condições reutilizáveis: na primeira utilização define-se a condição com todos os atributos e sub-elementos. Nas próximas utilizações, preenche-se somente o atributo PolicyConditionName, especificando o mesmo nome utilizado na primeira utilização da condição.

A Figura 6.13 apresenta um exemplo de condição reutilizável. Nesse exemplo, a condição "Condicao1" é definida dentro da regra "Regra1". Essa condição contém um filtro que é satisfeito para tráfego destinado à porta 80 do equipamento cujo IP é "10.0.0.1". Na regra "Regra2" a mesma condição é necessária, portanto, basta especificar o mesmo nome da condição utilizada na "Regra1".

```
<IPsecPolicyGroup PolicyGroupName="PolíticaExemplo">
   <IKERule PolicyRuleName="Regra1">
      <SACondition PolicyConditionName="Condicao1">
         <FilterList>
            <IPHeadersFilter HdrDestAddress="10.0.0.1"
               HdrDestPortStart="80"
               HdrDestPortEnd="80"/>
         </FilterList>
      </SACondition>
      <IKEAction>
         ...
      </IKEAction>
   </IKERule>
   <IPsecRule PolicyRuleName="Regra2">
      <SACondition PolicyConditionName="Condicao1"/>
      <IPsecAction>
         ...
      </IPsecAction>
   </IPsecRule>
</IPsecPolicyGroup>
```

Figura 6.13: Exemplo de Condição Reutilizável

Uma condição pode conter também um conjunto de entidades que gerenciam certificados que são aceitos. Se um dispositivo IPsec apresentar um certificado dessas entidades, então o certificado é aceito. Para representar as entidades, foi definido o elemento

AcceptCredentialsFrom (Figura 6.14), que possui um conjunto de autoridades certificadoras (representadas pelo elemento CertificateAuthority) e/ou um conjunto de nomes de servidores Kerberos (representados pelo elemento KerberosKeyDistributionCenter). A classe AcceptCredentialsFrom foi a única classe associativa para a qual foi criado um elemento no esquema XML. Isso foi feito para agrupar as entidades certificadoras, e facilitar a compreensão do esquema.

O elemento CertificateAuthority possui somente o atributo CADistinguishedName, que representa o nome distinto (*Distinguished Name – DN*) da autoridade certificadora.

Da mesma forma, o elemento KerberosKeyDistributionCenter possui somente o atributo Name, que representa o nome do servidor Kerberos.

```
<xsd:element name="AcceptCredentialsFrom" minOccurs="0">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="CertificateAuthority" minOccurs="0"
            maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:attribute name="CADistinguishedName"/>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="KerberosKeyDistributionCenter" minOccurs="0"
            maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:attribute name="Name"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Figura 6.14: Esquema XML do Elemento AcceptCredentialsFrom

6.3.5. Ações

As regras possuem um conjunto de ações, que devem ser executadas quando suas condições forem satisfeitas. As ações são representadas pelo elemento IKEAction, IKEReject, IPsecTransportAction, IPsecTunnelAction, IPsecBypassAction, IPsecDiscardAction e PreconfiguredSAAction.

O elemento IKERule pode conter os elementos IKEAction, IKEReject e PreconfiguredSAAction. O elemento IPsecRule pode conter os elementos IPsecAction, IPsecBypass, IPsecDiscard e PreconfiguredSAAction.

Da mesma forma que as condições, as ações também são divididas em dois grupos: específicas a uma regra e as reutilizáveis. O mecanismo é o mesmo utilizado para as

condições. Quando for reutilizar uma ação basta preencher o atributo PolicyActionName com o mesmo valor de uma ação definida anteriormente.

O elemento IKEAction (Figura 6.15) representa as configurações que serão utilizadas na negociação da fase 1 do IKE. Ele contém um ou mais elementos IKEProposal, que armazenam as informações sobre as propostas.

```
<xsd:complexType name="IKEActionType">
   <xsd:sequence>
      <xsd:element name="IKEProposal" type="IKEProposalType" minOccurs="0"
         maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="ExchangeMode" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="UseIKEIdentityType" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeKilobytes"
      type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="IdleDurationSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="AgressiveModeGroupId"
      type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.15: Esquema XML do Elemento IKEAction

Os elementos IPsecTransportAction (Figura 6.16) e IPsecTunnelAction (Figura 6.17) representam as configurações que serão utilizadas na negociação da fase 2 do IKE. Eles contêm um ou mais elementos IPsecProposal, que possuem as informações sobre as propostas para a fase 2.

O elemento IPsecTransportAction indica que deve ser utilizado o modo transporte. O elemento IPsecTunnelAction indica que deve ser utilizado o método túnel. Para indicar o dispositivo com o qual será criado o túnel, utiliza-se o elemento PeerGateway, que representa um computador ou *firewall*.

```
<xsd:complexType name="IPsecTransportActionType">
   <xsd:sequence>
      <xsd:element name="IPsecProposal" type="IPsecProposalType"
         minOccurs="0"   maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="UsePFS" type="xsd:boolean"/>
   <xsd:attribute name="UseIKEGroup" type="xsd:boolean"/>
   <xsd:attribute name="GroupId" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="Granularity" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="IdleDurationSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.16: Esquema XML do Elemento IPsecTransportAction

A classe associativa PeerGatewayForTunnel não foi representada diretamente no esquema XML. O atributo SequenceNumber, pertencente a essa classe e utilizado para indicar a ordem de precedência dos *gateways*, foi mapeado para o elemento PeerGateway.

```
<xsd:complexType name="IPsecTunnelActionType">
   <xsd:sequence>
      <xsd:element name="IPsecProposal" type="IPsecProposalType"
         minOccurs="0"   maxOccurs="unbounded"/>
      <xsd:element name="PeerGateway" minOccurs="1" maxOccurs="unbounded">
         <xsd:complexType>
            <xsd:attribute name="Name" type="xsd:string"/>
            <xsd:attribute name="PeerIdentityType"
               type="xsd:nonNegativeInteger"/>
            <xsd:attribute name="PeerIdentity" type="xsd:string"/>
            <xsd:attribute name="SequenceNumber"
               type="xsd:nonNegativeInteger"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="UsePFS" type="xsd:boolean"/>
   <xsd:attribute name="UseIKEGroup" type="xsd:boolean"/>
   <xsd:attribute name="GroupId" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="Granularity" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="IdleDurationSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="DFHandling" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.17: Esquema XML do Elemento IPsecTunnelAction

Além das classes de negociação do IPsec, existem ainda as classes chamadas estáticas, pois não requerem negociação. Essas classes já contêm os valores que seriam obtidos na negociação ou então bloqueiem a comunicação. São elas (Figura 6.18):

- IKERejectAction: É utilizada para prevenir negociações da primeira fase do IKE. O principal uso desta classe é prevenir alguns tipos de ataque por negação de serviço. O dispositivo IPsec deve simplesmente descartar os pacotes direcionados para o IKE (porta UDP 500).
- IPsecBypassAction: É utilizada para indicar que os pacotes podem ser enviados ou recebidos sem processá-los com o IPsec.
- IPsecDiscardAction: É utilizada para indicar que os pacotes devem ser descartados. É o mesmo que dizer que os pacotes foram negados.
- PreconfiguredSAAction: Essa classe é utilizada para indicar a construção de uma associação de segurança utilizando protocolos e chaves pré-configuradas.

```
<xsd:complexType name="IKERejectActionType">
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="LifetimeSeconds" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IPsecBypassActionType">
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="LifetimeSeconds" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IPsecDiscardActionType">
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="LifetimeSeconds" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="PreconfiguredSAActionType">
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="SPI" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.18: Esquema XML dos Elementos de Ações Estáticas

Os atributos DoActionLogging e DoPacketLogging foram herdados da classe SAAction. A classe associativa SAActionInRule não tem um mapeamento direto, portanto o atributo ActionOrder foi mapeado para as classes de ações.

O elemento PreconfiguredSAAction representa uma ação pré-configurada. Ou seja, não é necessário realizar a negociação para determinar quais algoritmos serão utilizados. Os algoritmos já estão definidos, inclusive as chaves e demais parâmetros.

O modelo IETF é genérico e permite representar quais são essas configurações e parâmetros. No caso do modelo XML proposto, que tem como objetivo definir políticas de segurança para *Web Services*, é necessário somente informar o número do SPI (que pode ser entendido como um contrato) da ação pré-configurada que será utilizada.

Antes de realizar a primeira chamada ao *Web Service*, as duas organizações farão um contrato determinando todos os parâmetros e protocolos a serem utilizados (esse contrato será firmado no mundo real, ou seja, as informações não trafegarão pela Internet). Nesse contrato também já estarão definidos os números dos SPI's que as configurações devem ter. No momento da chamada do serviço, o cliente obtém esse número e ativa a configuração correspondente.

Pelo fato de ser necessário um contrato diferente para cada organização cliente, a organização que estiver provendo os serviços deverá ter um mecanismo para diferenciar quem é o cliente e assim determinar o documento WSDL e a política de segurança que serão fornecidos a ele. Esse mecanismo é um trabalho futuro.

Os atributos do elemento PreconfiguredSAAction são:

- SPI: Especifica o valor do SPI (*Security Parameter Index*) da ação pré-configurada a ser utilizada. Esse número é o determinado pelo contrato firmado entre o cliente e o servidor, e as políticas devem estar configuradas pelos dois antes de qualquer chamada do *Web Service*.
- ActionOrder: Especifica a posição relativa da ação na seqüência de ações associadas à regra. Valores menores têm precedência sobre valores maiores.

Para os casos em que uma ação é composta por várias ações, devem ser utilizados os elementos CompoundIKEActionType (Figura 6.19) e CompoundIPsecActionType (Figura 6.20).

```
<xsd:complexType name="CompoundIKEActionType">
   <xsd:sequence>
      <xsd:element name="IKEAction" type="IKEActionType" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="IKERejectAction" type="IKERejectActionType"
        minOccurs="0"/>
      <xsd:element name="PreconfiguredSAAction" type="PreconfiguredSAActionType"
        minOccurs="0"/>
      <xsd:element name="CompoundIKEAction" type="CompoundIKEActionType"
        minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="SequencedActions" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="ExecutionStrategy" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.19: Esquema XML do Elemento CompoundIKEAction

```
<xsd:complexType name="CompoundIPsecActionType">
   <xsd:sequence>
      <xsd:element name="IPsecTransportAction" type="IPsecTransportActionType"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="IPsecTunnelAction" type="IPsecTunnelActionType"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="IPsecBypassAction" type="IPsecBypassActionType"
        minOccurs="0"/>
      <xsd:element name="IPsecDiscardAction" type="IPsecDiscardActionType"
        minOccurs="0"/>
      <xsd:element name="PreconfiguredSAAction" type="PreconfiguredSAActionType"
        minOccurs="0"/>
      <xsd:element name="CompoundIPsecAction" type="CompoundIPsecActionType"
        minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="SequencedActions" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="ExecutionStrategy" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.20: Esquema XML do Elemento CompoundIPsecAction

A Figura 6.21 apresenta um exemplo de utilização de ações compostas. O objetivo das ações deste exemplo é criar um canal IPsec no modo transporte com um servidor. Para esse canal poder ser criado, é necessário antes criar um canal IPsec no modo túnel com o dispositivo "Gateway1" ou com o dispositivo "Gateway2", através do qual o canal no modo transporte será criado.

Para representar essa política, é definida uma ação composta chamada "Composta2" (destacado pelo quadro tracejado), que contém as ações necessárias para criar um túnel com "Gateway1" ou com "Gateway2". Como o atributo ExecutionStrategy dessa ação composta é 2, o canal com Gateway2 somente será criado se a tentativa de criar o canal com Gateway1 falhar.

É definida também uma ação composta chamada "Composta1", que contém a ação "Composta2" e a ação necessária para criar o canal no modo transporte.

```
<CompoundIPsecAction PolicyActionName="Composta1" ExecutionStrategy="1">
  <CompoundIPsecAction PolicyActionName="Composta2" ExecutionStrategy="2">
    <IPsecTunnelAction PolicyActionName="TunelComGateway1">
      <IPsecProposal ...>
        <AHTransform ...>
      </IPsecProposal>
      <PeerGateway Name="Gateway1"/>
    </IPsecTunnelAction PolicyActionName="TunelComGateway2">
    <IPsecTunnelAction>
      <IPsecProposal ...>
        <AHTransform ...>
      </IPsecProposal>
      <PeerGateway Name="Gateway2"/>
    </IPsecTunnelAction>
  </CompoundIPsecAction>
  <IPsecTransportAction PolicyActionName="Transporte">
    <IPsecProposal ...>
      <ESPTransform ...>
    </IPsecProposal>
  </IPsecTransportAction>
</CompoundIPsecAction>
```

Figura 6.21: Exemplo de Utilização de Ações Compostas

6.3.6. Propostas e Transformações

As ações de negociação são compostas por uma ou mais propostas. O modelo IETF define dois tipos de propostas: IKEProposal (Figura 6.22), que representa uma proposta de negociação da fase 1 do IKE, e IPsecProposal, que representa uma proposta de negociação para a fase 2. As duas classes foram mapeadas para o esquema XML para dois elementos com o mesmo nome.

```
<xsd:complexType name="IKEProposalType">
  <xsd:attribute name="Name" type="xsd:string"/>
  <xsd:attribute name="CipherAlgorithm" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="HashAlgorithm" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="PRFAlgorithm" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="GroupId" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="AuthenticationMethod" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="MaxLifetimeSeconds" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="MaxLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="VendorID" type="xsd:string"/>
  <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.22: Esquema XML do Elemento IKEProposal

O IPsecProposal (Figura 6.23) contém um conjunto de transformações, sendo que cada transformação representa um conjunto de parâmetros para o protocolos AH, ESP ou IPComp. Cada classe de transformação foi mapeada para um elemento de mesmo nome.

```
<xsd:complexType name="IPsecProposalType">
   <xsd:sequence>
      <xsd:element name="AHTransform" type="AHTransformType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="ESPTransform" type="ESPTransformType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="IPCOMPTransform" type="IPCOMPTransformType"
         minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="Name" type="xsd:string"/>
   <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.23: Esquema XML do Elemento IPsecProposal

O elemento AHTransform (Figura 6.24) provê os parâmetros necessários para a negociação do protocolo AH.

Por ser uma classe associativa, a classe ContainedTransform não foi mapeada diretamente para o esquema XML. O atributo SequenceNumber dessa classe foi então inserido nos elementos de transformação. Esse atributo especifica a ordem de preferência das transformações.

```
<xsd:complexType name="AHTransformType">
   <xsd:attribute name="CommonName"/>
   <xsd:attribute name="AHTransformID" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MaxLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MaxLifetimeKilobytes"
      type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="UseReplayPrevention" type="xsd:boolean"/>
   <xsd:attribute name="ReplayPreventionWindowSize"
      type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.24: Esquema XML do Elemento AHTransform

O elemento ESPTransform (Figura 6.25) provê os parâmetros necessários para a negociação do protocolo ESP (utilizado para especificar parâmetros de criptografia e/ou autenticação).

```
<xsd:complexType name="ESPTransformType">
   <xsd:attribute name="CommonName"/>
   <xsd:attribute name="MaxLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MaxLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="IntegrityTransformId" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="CipherTransformId" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="CipherKeyLength" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="CipherKeyRounds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="UseReplayPrevention" type="xsd:boolean"/>
   <xsd:attribute name="ReplayPreventionWindowSize" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.25: Esquema XML do Elemento ESPTransform

O elemento IPCOMPTransform (Figura 6.26) provê os parâmetros necessários para a negociação do protocolo IPComp (utilizado para especificar os parâmetros para compressão de dados).

```
<xsd:complexType name="IPCOMPTransformType">
   <xsd:attribute name="CommonName"/>
   <xsd:attribute name="MaxLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MaxLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="Algorithm" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="DictionarySize" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="PrivateAlgorithm" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

Figura 6.26: Esquema XML do Elemento IPCOMPTransform

6.4. Extenção do WSDL para Representação de Políticas de Segurança

A especificação do WSDL permite que sejam criadas extensões da linguagem sem a necessidade de modificar a especificação [CHR01]. Essas extensões vão desde os tipos de dados até os protocolos que podem ser utilizados.

Após descrever em um arquivo XML as definições de segurança, será necessário acrescentar na seção *port* da seção *service* do arquivo de descrição WSDL um atributo chamado securityPolicy, que indicará o nome da política de segurança. A descrição da política de segurança, representada pelo elemento SecurityPolicy, deverá ser acrescentada como um elemento filho ao elemento description do arquivo WSDL.

Essas são as únicas modificações necessárias no arquivo WSDL original. A Figura 6.27 apresenta o esqueleto de um arquivo WSDL com a descrição da política de segurança. As alterações necessárias estão destacadas.

```
<?xml version="1.0"?>
<definitions name="StockQuote"  ...>
   <types>
   </types>

  <message name="...">
  </message>

  <portType name="...">
    <operation name="...">
    </operation>
  </portType>

  <binding name="StockQuoteSoapBinding"
          type="tns:StockQuotePortType">
  </binding>

   <service name="StockQuoteSevice">
      <documentation>My First Service</documentation>
      <port name="StockQuotePort" binding="tns:StockQuoteBinding"
            securityPolicy="StockQuoteSecurityPolicy">
        <SOAP:Address location="http://www.example/"/>
      </port>
   </service>

   <SecurityPolicy Name="StockQuoteSecurityPolicy">
      ...
   </SecurityPolicy>
</definition>
```

Figura 6.27: WSDL com Políticas de Segurança

6.5. Exemplo de Utilização do Modelo

A Figura 6.28 apresenta um exemplo de política de segurança utilizando o modelo proposto. A política representada nesse exemplo é a mesma da seção 4.5.

Foi criado um grupo de Políticas chamado PolíticaExemplo que contém duas regras, uma para a primeira fase da negociação IKE e a outra para a segunda fase. Note que nesse exemplo foram utilizadas condições reutilizáveis.

A primeira regra (Regra1) define duas condições, uma para a porta 80 (protocolo HTTP) e outra para a porta 25. A segunda regra reutiliza as condições da primeira regra, pois informa os mesmos nomes de condição.

O dispositivo deverá criar um canal IPsec para o tráfego que satisfaça essas condições. Para a primeira fase, deverão ser utilizados os protocolos DES-CBC para criptografia, MD5 para algoritmo de *hash* e assinaturas digitais RSA para autenticação (os protocolos e algoritmos são representados por números, definidos pelo IPsec DOI [PIP98]). Para a segunda fase, deverão ser utilizados os protocolos MD5 para o AH, SHA-1 para integridade do ESP e 3DES para criptografia do ESP.

```
<definitions ...>

   ... Definições do WSDL

   <SecurityPolicy Name="Exemplo">
      <IPsecPolicyGroup PolicyGroupName="PolíticaExemplo">
         <IKERule PolicyRuleName="Regra1">
            <SACondition PolicyConditionName="Condicao1">
               <FilterList>
                  <IPHeadersFilter HdrDestAddress="10.0.0.1"
                     HdrDestPortStart="80" HdrDestPortEnd="80"/>
               </FilterList>
            </SACondition>
            <SACondition PolicyConditionName="Condicao2">
               <FilterList>
                  <IPHeadersFilter HdrDestAddress="10.0.0.1"
                     HdrDestPortStart="25" HdrDestPortEnd="25"/>
               </FilterList>
            </SACondition>
            <IKEAction>
               <IKEProposal CipherAlgorithm="1" HashAlgorithm="1"
                  AuthenticationMethod="3"/>
            </IKEAction>
         </IKERule>
         <IPsecRule PolicyRuleName="Regra2">
            <SACondition PolicyConditionName="Condicao1"/>
            <SACondition PolicyConditionName="Condicao2"/>
            <IPsecTransportAction>
               <IPsecProposal>
                  <AHTransform AHTransformID="2" />
                  <ESPTransform IntegrityTransformId="2" CipherTransformId="3"/>
                  </IPsecProposal>
            </IPsecAction>
         </IPsecRule>
      </IPsecPolicyGroup>
   </SecurityPolicy>
</definitions>
```

Figura 6.28: Exemplo de Política

6.6. Requisitos Mínimos

Após obter o documento que descreve as políticas de segurança para acesso a um determinado *Web Service*, o cliente pode verificar se essas políticas atendem a requisitos mínimos. Esses requisitos mínimos podem ser protocolos, algoritmos, tamanhos de chave, autoridades certificadoras e outros parâmetros aceitos pelo cliente. Dessa forma, o cliente pode selecionar e configurar seu dispositivo IPsec somente com os parâmetros permitidos pela sua política de segurança.

O documento de requisitos mínimos pode conter várias combinações de protocolos e algoritmos aceitos pelo cliente. Mas, para simplificar a negociação do IPsec no momento da criação do canal, esse documento de requisitos mínimos deve ser construído de forma que o resultado de sua aplicação seja apenas uma combinação de protocolos e algoritmos. A escolha dessa combinação será a que tiver maior prioridade e que também

existir na política fornecida pelo *Web Service*. Fica a cargo do cliente determinar as prioridades, que podem ser determinadas por grau de segurança, desempenho ou outro critério.

O documento de requisitos mínimos é um documento representado na linguagem XSL [ADL01]. A Figura 6.29 apresenta um trecho do documento de Requisitos Mínimos. Esse documento foi construído de forma que todas as definições do WSDL sejam copiadas para o destino. No que se refere às políticas, esse documento copia todas as regras e condições. Somente as ações desejadas são copiadas. Os requerimentos especificados nesse documento são (aparecem destacados na figura):

- Utilizar o algoritmo 3DES (representado pelo número 5) na primeira fase do IKE;
- Utilizar o algoritmo de integridade SHA (representado pelo número 3) para o AH (modo transporte ou túnel);
- Utilizar algoritmo de criptografia 3DES (representado pelo número 3) para o protocolo ESP (modo transporte ou túnel).

O documento de Requisitos Mínimos completo pode ser encontrado no Anexo E.

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/xhtml1/strict">

   ...
   <xsl:template match="IKEAction">
      <xsl:text>
           </xsl:text>
      <xsl:copy>
      <xsl:for-each select="@*">
           <xsl:copy/>
      </xsl:for-each>
      <xsl:for-each select="*[self::IKEProposal]">
         <xsl:if test="@CipherAlgorithm='5'">
            <xsl:text>
               </xsl:text>
               <xsl:copy-of select="."/>
            </xsl:if>
      </xsl:for-each>
      <xsl:text>
           </xsl:text>
      </xsl:copy>
   </xsl:template>

   <xsl:template match="IPsecTransportAction">
      <xsl:text>
           </xsl:text>
      <xsl:copy>
      <xsl:for-each select="@*">
           <xsl:copy/>
      </xsl:for-each>
      <xsl:for-each select="*">
         <xsl:if test="AHTransform/@AHTransformID='3' and
            ESPTransform/@CipherTransformId='3'">
            <xsl:text>
               </xsl:text>
               <xsl:copy-of select="."/>
            </xsl:if>
      </xsl:for-each>
      <xsl:text>
           </xsl:text>
      </xsl:copy>
   </xsl:template>
   ...
</xsl:stylesheet>
```

Figura 6.29: Trecho de um Documento de Requisitos Mínimos

6.7. API

Este trabalho propõe uma API para trabalhar com a configuração das políticas IPsec pelo cliente.

Essa API foi desenvolvida utilizando a plataforma Java 2 Standard Edition 1.4.1 da SUN, e o pacote de desenvolvimento JWSDP (Java Web Services Developer Pack) 1.1 da mesma fornecedora.

A API é composta pela classe SecurityManager que contém 6 métodos:

- SecurityManager (método construtor da classe);

- loadPolicyFile;
- loadPolicyFromURL;
- applyPolicy;
- removePolicy;
- verifyIPsecChannel.

Esta API tem o objetivo de simplificar o desenvolvimento de aplicações clientes de *Web Services*, pois o desenvolvedor não precisa se preocupar com os detalhes de obtenção e configuração das políticas no computador. O desenvolvedor realiza algumas chamadas para esta API, e ela se encarrega de obter as políticas, aplicá-las e por fim removê-las.

Para permitir a portabilidade da API, além da utilização da plataforma Java também foi utilizada a linguagem XSL [ADL01], que é uma linguagem para transformação de documentos XML em outros documentos (XML ou não). O documento XSL é utilizado para transformar as políticas de segurança em comandos, de forma que a execução desses comandos realize a configuração do IPsec. Assim, para permitir a utilização da API em outros sistemas operacionais ou com outras implementações do IPsec, basta modificar o XSL de acordo com a necessidade. Da mesma forma outro documento XSL é utilizado para criar os comandos de remoção das políticas.

O arquivo de requisitos mínimos também é um documento de transformação XSL. Esse arquivo contém as informações sobre os requisitos mínimos aceitáveis, principalmente os protocolos, algoritmos e tamanhos de chave. Esses requisitos podem ser de segurança, desempenho ou outra característica, dependendo da necessidade e da política da organização.

Através das informações contidas no arquivo de requisitos mínimos, a API extrai somente os protocolos e combinações de parâmetros preferíveis (maior segurança, maior desempenho, etc). Isso significa que no momento da negociação, o dispositivo IPsec terá somente uma proposta a fornecer para o servidor, que será aceita pois ela estava especificada na política de segurança do mesmo. Isso irá diminuir o impacto no desempenho da chamada do *Web Service* causado pela negociação do IKE.

Para cada servidor que o cliente desejar acessar ele deverá criar um objeto da classe SecurityManager. Cada objeto conterá as informações sobre as políticas de um determinado servidor, de forma que quando o cliente desejar realizar chamadas para um determinado *Web Service*, basta selecionar o objeto correspondente e chamar os métodos de aplicação e remoção de políticas.

O código fonte completo da API é apresentado no Anexo C.

6.7.1. Construindo uma instância da Classe SecurityManager

Método	SecurityManager
Descrição	Construtor da classe SecurityManager.
Parâmetros	• ApplyPolicyXSLFileName - Nome do arquivo contendo o documento XSL para converter o documento de políticas em comandos de aplicação das políticas para o dispositivo IPsec. • RemovePolicyXSLFileName - Nome do arquivo contendo o documento XSL para converter o documento de políticas em comandos de remoção das políticas para o dispositivo IPsec. • RequirementsXSLFileName - Nome do arquivo contendo os requisitos mínimos.
Retorno	Nenhum

Esse método é o construtor da classe SecurityManager. Os parâmetros são os nomes dos arquivos contendo os documentos de transformação XSL. Esses arquivos somente serão utilizados no momento em que as políticas forem carregadas e avaliadas.

O cliente tem a opção de não informar o nome do arquivo que contém os requisitos mínimos. Caso isso ocorra, a API entenderá que qualquer política pode ser aplicada.

6.7.2. Carregando e Avaliando as Políticas

Método	loadPolicyFile (para arquivo local) loadPolicyFromURL (para endereço da Internet)
Descrição	Carrega a política a partir do documento WSDL (de um arquivo ou de um endereço da Internet) e faz a verificação se ela atende os requisitos mínimos de segurança estipulados pelo cliente. Se a política atender aos requisitos mínimos, separa o conjunto de protocolos preferível.
Parâmetros	• PolicyFileName ou URL - Nome do arquivo ou endereço Internet do documento WSDL.

Retorno	Os valores possíveis são: • SecurityPolicy.Ok – A política foi aceita e está pronta para ser aplicada. • SecurityPolicy.NotAccepted – A política não atende os requisitos mínimos. • SecurityPolicy.FileNotFound – Arquivo de políticas não encontrado. • SecurityPolicy.URLNotFound – O documento de políticas não foi encontrado no endereço informado. • SecurityPoilcy.PolicyXMLError – O documento que descreve as políticas tem algum erro de sintaxe. • SecurityPolicy.TransformXMLError – O documento XSL de transformação tem algum erro de sintaxe.

As políticas deverão estar em um documento WSDL, que pode estar em um arquivo local ou então estar disponível através da Internet. A API permite buscar este documento tanto localmente (através da indicação da localização do arquivo) quanto da Internet (através do endereço fornecido).

Os documentos XSL são utilizados por esse método para verificar os requisitos mínimos (caso tenha sido informado) e extrair os comandos de aplicação e remoção das políticas.

Esse método pode retornar os seguintes erros:

- Política não possui informações sobre os algoritmos desejados;
- Política não atende aos requisitos mínimos;
- Arquivo WSDL não encontrado;
- Endereço da Internet não encontrado;
- Documento de descrição da política de segurança possui erro de sintaxe;
- Documento de transformação XSL possui erro de sintaxe.

Se ocorrer qualquer um desses erros, não será possível aplicar a política, portanto a chamada ao *Web Service* não terá sucesso.

6.7.3. Aplicação das Políticas

Método	applyPolicy
Descrição	Configura o dispositivo IPsec com as políticas de segurança.
Parâmetros	Nenhum
Retorno	Os valores possíveis são: • SecurityPolicy.Ok – a política foi aplicada corretamente. • SecurityPolicy.Failure – houve algum problema na aplicação da política. • SecurityPolicy.ExecNotFound – o programa executável necessário para aplicar a política não foi encontrado.

A aplicação das políticas se refere a configurar o dispositivo IPsec de acordo com as políticas obtidas do documento WSDL. Nesse momento o canal ainda não é criado. O dispositivo IPsec irá automaticamente criar o canal no momento em que houver a chamada do *Web Service*. Quando o canal IPsec não for mais necessário, o método de remoção de políticas irá retirar as configurações realizadas pelo método applyPolicy.

Esse método pode retornar os seguintes erros:

- Falha na aplicação da política: a política de segurança obtida contém erros, como por exemplo, um endereço IP inválido (300.1.1.400).
- Executável não encontrado: Para aplicar as políticas, a API utiliza um comando do sistema operacional. Esse aplicativo deve estar disponível no momento da aplicação da política. O nome do aplicativo varia de acordo com o sistema operacional.

6.7.4. Remoção das Políticas

Método	removePolicy
Descrição	Remove as configurações de segurança aplicadas pelo método applyPolicy.
Parâmetros	Nenhum

Retorno	Os valores possíveis são: • SecurityPolicy.Ok – a política foi removida com sucesso. • SecurityPolicy.Failure – houve algum problema na remoção da política. • SecurityPolicy.ExecNotFound – o programa executável necessário para remover a política não foi encontrado.

Este método removerá as políticas aplicadas pelo método applyPolicy. Como as políticas foram aplicadas de forma dinâmica, após a remoção não ficará nenhuma informação sobre elas no registro do sistema operacional.

Caso seja necessário reaplicar as políticas, basta chamar novamente o método applyPolicy.

Esse método retorna os seguintes valores:

- Ok: indica que a remoção das políticas ocorreu com sucesso.
- Falha na execução do comando: o comando retornou alguma mensagem de erro.
- Executável não encontrado: da mesma forma que para aplicar a política, é necessário o um aplicativo para removê-la. Esse erro será retornado quando esse aplicativo não for encontrado.

6.7.5. Verificação do Canal IPsec

Método	verifyIPsecChannel
Descrição	Verifica se o canal foi construído corretamente. Esse método deve ser chamado após a chamada de um serviço do *Web Service*, pois o IPsec só constrói o canal no momento em que ele é necessário.
Parâmetros	Nenhum
Retorno	Os valores possíveis são: • SecurityPolicy.Ok – nenhum problema reportado pelo dispositivo IPsec. • SecurityPolicy.NotAvailable – dispositivo IPsec não responde aos comandos. Pode estar desativado.

	• SecurityPolicy.ExecNotFound – o programa executável necessário para aplicar a política não foi encontrado. • SecurityPolicy.OnlyPhase1 – a política de segurança somente negociou a fase 1 do IKE. • SecurityPolicy.UnknowError – erro desconhecido. Não foi possível determinar a origem do erro.

O método verifyIPsecChannel irá verificar o estado do IPsec, para determinar se o canal foi criado corretamente. Como o canal somente é criado no momento da chamada do *Web Service*, é necessário que seja realizada pelo menos uma chamada antes de ser possível verificar o estado do canal.

É importante lembrar que esse método somente irá reportar problemas relacionados ao IPsec. Outros problemas, como rede indisponível ou servidor fora de funcionamento, não podem ser identificados.

Esse método retorna o estado do canal. O estado pode ser:

- Canal criado corretamente: Nesse caso, o canal IPsec foi criado corretamente. Se a chamada do *Web Service* não teve sucesso, a causa do problema pode estar no servidor ou no próprio *Web Service*.
- Falha na construção do canal: A falha na construção do canal pode ocorrer por diversas causas: desde impossibilidade de enviar os dados para o *Web Service* por falha de comunicação na Internet até por modificação das políticas do servidor não documentada.
- Sucesso apenas na negociação da fase 1 do IKE: Em alguns casos pode acontecer do dispositivo IPsec negociar apenas a primeira fase do IKE com sucesso. Um dos motivos que pode causar isso são regras para a segunda fase incorretas ou incompletas.
- Erro desconhecido: Não é possível determinar a origem do erro ou o estado do canal é desconhecido.

Os erros retornados pelo método são:

- Executável não encontrado: para determinar o estado do canal, este método depende de um comando do sistema operacional, e este deve estar disponível no momento da execução deste método.

6.7.6. Documento XSL para Aplicação de Políticas para Windows 2000

O documento de transformação é responsável por especificar as regras de conversão da política em uma seqüência de comandos a serem executados. A execução desses comandos irá aplicar a política.

Esse documento varia de acordo com o dispositivo IPsec, pois cada um tem seus próprios comandos e formas diferentes de configuração. Da Figura 6.31 até a Figura 6.40 são apresentados os trechos de um arquivo de transformação para ser utilizado com o Microsoft Windows 2000. A Figura 6.30 apresenta a sintaxe genérica do comando ipsecpol. O uso de seus parâmetros é explicado na seqüência.

```
ipsecpol    [\\computername] [-?] [-f FilterList] [-n NegotiationPolicyList]
            [-t TunnelAddress] [-a AuthMethodList] [-u] [-soft]
            [{-dialup | -lan}] [-ls SecurityMethodList] [-lk PhaselRekeyAfter]
            [-ip] [-confirm] [-w TYPE:DOMAIN] [-p PolicyName:PollInterval]
            [-r RuleName] [-x] [-y] [-o]
```

Figura 6.30: Sintaxe do Comando ipsecpol do MS Windows 2000

Pelo fato do comando ipsecpol e da própria implementação do IPsec do Windows não aceitar todos os parâmetros existentes no modelo, somente os parâmetros suportados são considerados no documento de transformação. O Anexo D contém o documento XSL completo.

A Figura 6.31 apresenta o trecho do documento XSL destinado a tratar os elementos definitions (WSDL), SecurityPolicy e IPsecPolicyGroup. O *template*[24] para o elemento definitions simplesmente trata o elemento SecurityPolicy. No tratamento do elemento SecurityPolicy é acrescentada uma mensagem "Aplicando Políticas", meramente informativa. O *template* do elemento IPsecPolicyGroup trata as regras IKE e IPsec separadamente, e no fim acrescenta uma instrução para tornar a política ativa.

[24] Os *templates* são modelos utilizados para indicar a transformação a ser realizada. Para cada elemento do documento de entrada a ser tratado deve haver um *template* que indica como será a representação no documento de saída para esse elemento.

```
<xsl:stylesheet version="1.0" ...>
   <xsl:template match="definitions">
      <xsl:apply-templates select="SecurityPolicy"/>
   </xsl:template>

   <xsl:template match="SecurityPolicy">
      <xsl:text>echo Aplicando politicas</xsl:text>
      <xsl:apply-templates select="IPsecPolicyGroup"/>
   </xsl:template>

   <xsl:template match="IPsecPolicyGroup">
echo Politica: <xsl:value-of select="@PolicyGroupName"/>
      <xsl:apply-templates select="IKERule"/>
      <xsl:apply-templates select="IPsecRule"/>
      <xsl:text>
ipsecpol</xsl:text>
      <xsl:text> -w REG -x -p </xsl:text>
         <xsl:value-of select="@PolicyGroupName"/>
   </xsl:template>
```

Figura 6.31: Política de Segurança e Grupos de Política

Os *templates* para os elementos IKERule e IPsecRule são apresentados na Figura 6.32 e na Figura 6.33, respectivamente. Para cada regra será criado um comando ipsecpol. O nome da política e o nome da regra são acrescentados após as chaves "-p" e "-r". As condições devem ser aplicadas após a chave de comando "-f", que aparece logo após o comando ipsecpol e antes da chamada do template de tratamento de Condições. Após tratar as condições, se a regra for IKE então a ação IKE (IKEAction) deve ser tratada. Se a regra for IPsec, então as ações IPsec (IPsecTransportAction e IPsecTunnelAction) serão tratadas.

```
   <xsl:template match="IKERule">
  echo Regra: <xsl:value-of select="@Name"/>
  <xsl:text>
  ipsecpol</xsl:text>
      <xsl:text> -w REG -p </xsl:text>
      <xsl:value-of select="../@PolicyGroupName"/>
      <xsl:text> -r </xsl:text> <xsl:value-of select="@PolicyRuleName"/>
      <xsl:text> -f</xsl:text>
      <xsl:apply-templates select="SACondition"/>

      <xsl:for-each select="IKEAction">
         <xsl:apply-templates select="."/>
         <xsl:if test="@PolicyActionName">
            <xsl:variable name="ActionName" select="@PolicyActionName"/>
            <xsl:variable name="RuleName" select="../@PolicyRuleName"/>
            <xsl:apply-templates select="//IKEAction[attribute::
               PolicyActionName=$ActionName and
               ../@PolicyRuleName!=$RuleName]"/>
         </xsl:if>
      </xsl:for-each>
   </xsl:template>
```

Figura 6.32: Transformação das Regras IKE (1ª. Fase)

```
<xsl:template match="IPsecRule">
echo Regra: <xsl:value-of select="@Name"/>
   <xsl:text>
ipsecpol</xsl:text>
   <xsl:text> -w REG -p </xsl:text>
   <xsl:value-of select="../@PolicyGroupName"/>
   <xsl:text> -r </xsl:text> <xsl:value-of select="@PolicyRuleName"/>
   <xsl:text> -f</xsl:text>
   <xsl:apply-templates select="SACondition"/>
   <xsl:text> -n </xsl:text>
   <xsl:for-each select="IPsecTransportAction">
      <xsl:apply-templates select="."/>
      <xsl:if test="@PolicyActionName">
         <xsl:variable name="ActionName" select="@PolicyActionName"/>
         <xsl:variable name="Rule" select=".."/>
         <xsl:apply-templates select="//IPsecTransportAction[attribute::
            PolicyActionName=$ActionName and ..!=$Rule]"/>
      </xsl:if>
   </xsl:for-each>
   <xsl:for-each select="IPsecTunnelAction">
         <xsl:apply-templates select="."/>
      <xsl:if test="@PolicyActionName">
         <xsl:variable name="ActionName" select="@PolicyActionName"/>
         <xsl:variable name="Rule" select=".."/>
         <xsl:apply-templates select="//IPsecTunnelAction[attribute::
            PolicyActionName=$ActionName and ..!=$Rule]"/>
      </xsl:if>
   </xsl:for-each>
</xsl:template>
```

Figura 6.33: Transformação das Regras IPsec (2ª. Fase)

A Figura 6.34 apresenta os *templates* para tratamento das condições. O tratamento do elemento de condição é converter os filtros para o formato esperado pelo comando ipsecpol. Como a condição pode ser do tipo reutilizável, é necessário procurar em outras regras se existe uma condição com o mesmo nome. Se existir, os filtros da condição encontrada são utilizados.

```
<xsl:template match="SACondition">
   <xsl:apply-templates select="FilterList"/>
   <xsl:if test="@PolicyConditionName">
      <xsl:variable name="ConditionName" select="@PolicyConditionName"/>
      <xsl:variable name="Rule" select=".."/>
      <xsl:apply-templates select="//SACondition[attribute::
         PolicyConditionName=$ConditionName and ..!=$Rule]/FilterList"/>
   </xsl:if>
</xsl:template>
```

Figura 6.34: Transformação das Condições

O comando ipsecpol recebe os filtros representados pelos endereços de origem e destino, pelas portas de origem e destino e pelo protocolo do pacote. Os demais filtros não são aceitos pelo ipsecpol. O formato do filtro é o seguinte: "IPOrigem/Máscara:porta=IPDestino/Mascara:porta:protocolo". Para indicar que o filtro se

aplica nos dois sentidos, o sinal "=" é substituído pelo sinal "+". Esses dados são extraídos do elemento IPHeadersFilter, atributos HdrSrcAddress, HdrDestAddress, HdrSrcPortStart, HdrDestPortStart e HdrProtocolID, respectivamente (Figura 6.35).

```
<xsl:template match="FilterList">
  <xsl:for-each select="IPHeadersFilter">
    <xsl:text> </xsl:text>
    <xsl:value-of select="@HdrSrcAddress"/>
    <xsl:if test="@HdrSrcMask!=''">
      /<xsl:value-of select="@HdrSrcMask"/>
    </xsl:if>
    <xsl:if test="@HdrSrcPortStart!=''">
      :<xsl:value-of select="@HdrSrcPortStart"/>
    </xsl:if>
    <xsl:choose>
      <xsl:when test="../@Direction='4'">+</xsl:when>
      <xsl:otherwise>=</xsl:otherwise>
    </xsl:choose>
    <xsl:value-of select="@HdrDestAddress"/>
    <xsl:if test="@HdrDestMask!=''">
      /<xsl:value-of select="@HdrDestMask"/>
    </xsl:if>
    <xsl:if test="@HdrDestPortStart!=''">
      :<xsl:value-of select="@HdrDestPortStart"/>
    </xsl:if>
    <xsl:if test="@HdrDestPortStart!='' and HdrProtocolID!=''">
      :<xsl:value-of select="@HdrProtocolID"/>
    </xsl:if>
    <xsl:if test="@HdrDestPortStart='' and HdrProtocolID!=''">
      ::<xsl:value-of select="@HdrProtocolID"/>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

Figura 6.35: Transformação dos Filtros

O tratamento das ações da primeira fase do IKE é apresentado na Figura 6.36. A chave de comando "-a" é utilizada para indicar o algoritmo de autenticação. Para simplificar a figura, o tratamento desse algoritmo é apresentado na Figura 6.37. Logo após são tratados os tempos de vida, que aparecem com detalhes na Figura 6.38. A chave "-ls" indica o algoritmo de criptografia, o algoritmo de *hash* e o *groupId* da primeira fase da negociação, quem devem ser informados da seguinte forma: "AlgoritmoCriptografia-AlgoritmoHash-GroupId" (usando hífen para separar os algoritmos).

```
<xsl:template match="IKEAction">
   <xsl:if test="count(IKEProposal)>0">
      <xsl:text> -a </xsl:text>
      <xsl:for-each select="IKEProposal">
         <!-- Tratamento do método de autenticação (Figura 6.37) -->
         <xsl:if test="@usePFS='True'">
            <xsl:text> -lp</xsl:text>
         </xsl:if>
         <!-- Tratamento dos tempos de vida (Figura 6.38) -->
      </xsl:for-each>
      <xsl:text> -ls </xsl:text>
      <xsl:for-each select="IKEProposal">
         <xsl:choose>
            <xsl:when test="@CipherAlgorithm='1'">DES</xsl:when>
            <xsl:when test="@CipherAlgorithm='5'">3DES</xsl:when>
         </xsl:choose>
         <xsl:text>-</xsl:text>
         <xsl:choose>
            <xsl:when test="@HashAlgorithm='1'">MD5</xsl:when>
            <xsl:when test="@HashAlgorithm='2'">SHA</xsl:when>
         </xsl:choose>
         <xsl:text>-</xsl:text>
         <xsl:value-of select="@GroupId"/>
      </xsl:for-each>
   </xsl:if>
</xsl:template>
```

Figura 6.36: Transformação de Ações IKE

A Figura 6.37 apresenta o tratamento dos métodos de autenticação. Caso seja informado o método de autenticação com Certificados, é acrescentado o texto "CERT:" seguido da lista de autoridades certificadoras aceitas. Se for informado o método Kerberos, somente o texto "KERBEROS" é acrescentado. Se nenhum dos dois for utilizado, o documento XSL foi preparado para copiar o conteúdo informado do documento de políticas.

```
<xsl:choose>
   <xsl:when test="@AuthenticationMethod='3'">
      <xsl:for-each
         select="../../SACondition/AcceptCredentialsFrom/CertificateAuthority/">
         <xsl:text>CERT:"</xsl:text>
         <xsl:value-of select="@CADistinguishedName"/>
         <xsl:text>" </xsl:text>
      </xsl:for-each>
   </xsl:when>
   <xsl:when test="@AuthenticationMethod='6'">KERBEROS</xsl:when>
   <xsl:otherwise>
      <xsl:value-of select="@AuthenticationMethod"/>
   </xsl:otherwise>
</xsl:choose>
```

Figura 6.37: Tratamento de Métodos de Autenticação

A Figura 6.38 apresenta o tratamento dos tempos de vida da associação de segurança. A chave "-lp" é utilizada para indicar que a próxima informação está relacionada com os tempos de vida (pode ser informado apenas um ou ambos os tempos de vida). Um

número seguido da letra "S" indica o tempo de vida em segundos. Um número seguido da letra "Q" indica o tempo de vida em kbytes.

```
<xsl:if test="@MaxLifetimeSeconds!='' or @MaxLifetimeKilobytes!=''">
   <xsl:text> -lk </xsl:text>
</xsl:if>
<xsl:if test="@MaxLifetimeSeconds!=''">
   <xsl:value-of select="@MaxLifetimeSeconds"/>
   <xsl:text>S</xsl:text>
</xsl:if>
<xsl:if test="@MaxLifetimeSeconds!='' and @MaxLifetimeKilobytes!=''">
   <xsl:text>/</xsl:text>
</xsl:if>
<xsl:if test="@MaxLifetimeKilobytes!=''">
   <xsl:value-of select="@MaxLifetimeKilobytes"/>
   <xsl:text>Q</xsl:text>
</xsl:if>
```

Figura 6.38: Tratamento de Tempos de Vida

A Figura 6.39 apresenta o tratamento das ações da segunda fase do IKE, para o modo transporte. Os protocolos e algoritmos a serem utilizados na segunda fase são indicados da seguinte forma:

- AH[AlgoritmoIntegridade]: indica o uso do protocolo AH com o algoritmo de integridade especificado entre os colchetes.
- ESP[AlgoritmoCriptografia, AlgoritmoIntegridade] indica o uso do protocolo ESP com o algoritmo de criptografia e o algoritmo de integridade especificados entre os colchetes. Opcionalmente, se um dos algoritmos não for utilizado, deve-se utilizar a palavra "NONE" para indicar o algoritmo não utilizado.
- AH[AlgoritmoIntegridade]+ESP[AlgoritmoCriptografia, AlgoritmoIntegridade]: indica o uso dos protocolos AH e ESP.

```
<xsl:template match="IPsecTransportAction">
   <xsl:if test="count(IPsecProposal)>0">
      <xsl:for-each select="IPsecProposal">
         <xsl:sort select="@SequenceNumber"/>
         <xsl:if test="AHTransform[@AHTransformID]">
            <xsl:text>AH[</xsl:text>
               <xsl:choose>
                  <xsl:when test="AHTransform/@AHTransformID='2'">MD5</xsl:when>
                  <xsl:when test="AHTransform/@AHTransformID='3'">SHA</xsl:when>
               </xsl:choose>
            <xsl:text>]</xsl:text>
         </xsl:if>
         <xsl:if test="AHTransform and ESPTransform">
            <xsl:text>+</xsl:text>
         </xsl:if>
         <xsl:if test="ESPTransform">
            <xsl:text>ESP[</xsl:text>
            <xsl:choose>
               <xsl:when test="ESPTransform/@CipherTransformId='2'">DES</xsl:when>
               <xsl:when test="ESPTransform/@CipherTransformId='3'">3DES</xsl:when>
               <xsl:when test="not(ESPTransform/@CipherTransformId)">
                  NONE</xsl:when>
            </xsl:choose>
            <xsl:text>,</xsl:text>
            <xsl:choose>
               <xsl:when test="ESPTransform/@IntegrityTransformId='1'">
                  MD5</xsl:when>
               <xsl:when test="ESPTransform/@IntegrityTransformId='2'">
                  SHA</xsl:when>
               <xsl:when test="not(ESPTransform/@IntegrityTransformId)">
                  NONE</xsl:when>
            </xsl:choose>
            <xsl:text>]</xsl:text>
         </xsl:if>
      </xsl:for-each>
   </xsl:if>
</xsl:template>
```

Figura 6.39: Transformação de Ações IPsec Modo Transporte (IPsecTransportAction)

A Figura 6.40 apresenta o tratamento das ações da segunda fase do IKE para o modo túnel. Para o modo túnel, a forma de utilização é a mesma do modo transporte. A diferença é o acréscimo das informações sobre o *gateway* de segurança, indicado pela chave de comando "-t". Como a maior parte é semelhante ao tratamento do elemento IPsecTransportAction, os trechos semelhantes foram omitidos da figura.

```
<xsl:template match="IPsecTunnelAction">
   <xsl:for-each select="IPsecProposal">
      <!--Semelhante ao IPsecTransportAction-->
   </xsl:for-each>
   <xsl:apply-templates select="PeerGateway"/>
</xsl:template>

<xsl:template match="PeerGateway">
   <xsl:text> -t </xsl:text>
   <xsl:value-of select="PeerIdentity"/>
</xsl:template>
```

Figura 6.40: Transformação de Ações IPsec Modo Túnel (IPsecTunnelAction)

6.7.7. Documento XSL para Remoção de Políticas para Windows 2000

A Figura 6.41 apresenta o documento de transformação para criação dos comandos de remoção de Políticas. É um documento simples, cujo objetivo é obter o nome das políticas e criar o comando ipsecpol para remoção das mesmas. A chave de comando "-o" indica que a política indicada pela chave "-p" deve ser removida. A chave "-w REG" indica que a política está armazenada no registro do sistema.

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/xhtml1/strict">
   <xsl:output method="text" indent="no"/>
   <xsl:template match="definitions">
      <xsl:apply-templates select="SecurityPolicy"/>
   </xsl:template>

   <xsl:template match="SecurityPolicy">
      <xsl:text>echo Removendo politicas</xsl:text>
      <xsl:apply-templates select="IPsecPolicyGroup"/>
   </xsl:template>

   <xsl:template match="IPsecPolicyGroup">
echo Politica: <xsl:value-of select="@PolicyGroupName"/>
   <xsl:text>
ipsecpol</xsl:text>
      <xsl:text> -w REG -o -p </xsl:text>
         <xsl:value-of select="@PolicyGroupName"/>
   </xsl:template>
</xsl:stylesheet>
```

Figura 6.41: Documento XSL para transformação em Comandos de Remoção de Políticas

6.8. Conclusão do Capítulo

O presente capítulo apresentou o esquema XML proposto para o mapeamento das políticas IPsec, a API e o documento de requisitos mínimos.

Algumas classes não foram representadas no esquema XML pois representam estados internos de um dispositivo IPsec (como as classes de ações pré-configuradas).

Para permitir a aplicação das políticas definidas pelo modelo, foi desenvolvida uma API para interpretar e aplicar as políticas. Para que a API não ficasse limitada a um determinado dispositivo IPsec, a conversão das políticas para os comandos de configuração é realizada através de um documento de transformação XSL. Dessa forma, basta criar um documento XSL para o sistema operacional desejado e utilizar o mesmo código implementado em Java.

O mesmo procedimento foi adotado para representação dos requisitos mínimos de segurança. Os requisitos são representados na forma de um documento XSL, e podem ser facilmente modificados para representar qualquer tipo de requisitos.

É importante ressaltar que o modelo proposto representa a política de segurança entre o cliente e o servidor. Esse modelo não se propõe a representar diversas políticas, uma para cada trecho, quando há intermediários no caminho até o servidor. Supondo dois servidores, s1 e s2, e um cliente c1, o cliente c1 irá utilizar as políticas do servidor s1 quando desejar comunicar-se com ele. Se o servidor s1 for um intermediário, ele será responsável por buscar a política de segurança de s2 e aplicá-la. Esse processo será totalmente transparente para o cliente c1.

O capítulo seguinte apresenta um estudo de caso, no qual o servidor implementa e disponibiliza as políticas de segurança, e o cliente obtém o arquivo de políticas, aplica as políticas e executa chamadas ao *Web Service*. Essa implementação foi feita de forma a permitir a realização de testes de desempenho, com o objetivo de avaliar o impacto da utilização do IPsec nas chamadas para *Web Services*.

Capítulo 7

Estudo de Caso

7.1. Introdução ao Capítulo

Este capítulo apresenta um estudo de caso, utilizando a proposta descrita no capítulo anterior. O objetivo é comprovar o funcionamento e avaliar o impacto causado no desempenho dos *Web Services*. Sendo assim, foram implementados uma aplicação cliente e um *Web Service*.

Para avaliar o impacto dos diferentes protocolos e algoritmos, vários documentos WSDL foram criados, cada um contendo uma política com requisitos de segurança diferentes.

A seção 7.2 apresenta o cenário utilizado para avaliação. A seção 7.3 explica como os programas foram construídos e quais as suas características. A seção 7.4 apresenta os resultados obtidos. A seção 7.5 conclui o capítulo.

7.2. Cenário

O cenário proposto é composto por um *Web Service* e uma aplicação cliente. Para fins de avaliação, é possível executar tanto o cliente quanto o servidor utilizando ou não as políticas IPsec.

Quando o IPsec estiver sendo utilizado, o serviço disponibilizado pelo *Web Service* poderá ser acessado de acordo com as políticas de segurança fornecidas junto com o arquivo WSDL que descreve o serviço. Nesse caso, assume-se que essas políticas já estarão configuradas no servidor do fornecedor.

Para avaliação de desempenho, esse serviço é simplesmente um "ping". O *Web Service* recebe a informação na forma de uma cadeia de caracteres e a devolve da mesma forma.

O cliente foi implementado para permitir diversas chamadas ao *Web Service*, e possui mecanismos para permitir a mensuração do tempo gasto em cada chamada, além de registrar outros tempos, como por exemplo o tempo gasto para aplicação da política.

Para realização dos testes foram utilizados dois computadores, um Pentium III 900MHz como cliente e um AMD Duron 850MHz como servidor, os dois com 192MB de memória RAM. Foi utilizado um cabo simples para interligar os dois computadores, simulando uma rede de 100Mbps. Os dois equipamentos possuem o sistema operacional Microsoft Windows 2000 Professional instalado. O Servidor Apache Tomcat versão 4.1 foi utilizado como hospedeiro do *Web Service*, que acompanha o pacote de desenvolvimento JSWDP 1.1 (Java Web Services Developer Pack) da SUN.

Nos testes realizados o servidor estava dedicado ao *Web Service* e ao processamento IPsec (quando utilizado).

7.3. Implementação

A implementação foi realizada utilizando a plataforma Java 2 Standard Edition 1.4.1 da SUN, e o pacote de desenvolvimento JWSDP 1.1 da mesma fornecedora.

O programa foi implementado para realizar o seguinte fluxo:

1. Criar um objeto da classe SecurityManager;
2. Carregar as políticas a partir do WSDL;
3. Converter as políticas para comandos IPsec;
4. Aplicar a política IPsec (executar os comandos obtidos no item anterior);
5. *Loop* principal do programa (repetido 100 vezes):
    a. Obter o tempo atual (inicial) em milissegundos;
    b. Realizar 10 chamadas ao *Web Service*;
    c. Obter o tempo atual (final) em milissegundos;
    d. Armazenar o tempo gasto por chamada;
6. Remover a política.

7.4. Apresentação dos Resultados

O programa foi executado utilizando diversas combinações de protocolos e algoritmos de criptografia e *hashing* (conforme ilustra a Tabela 7.1). Para cada combinação a aplicação foi executada 10 vezes, obtendo no total 10.000 chamadas ao *Web Service* por combinação. O tempo na primeira chamada também foi registrado, pois nele está incluído o tempo gasto para a negociação do canal IPsec.

A Figura 7.1 apresenta a tela do programa utilizado para analisar os pacotes IP. A política de segurança utilizada no momento da captura especificava a utilização do protocolo AH utilizando como algoritmo de autenticação o MD5. Pode-se observar que a negociação realizada pela implementação IPsec do Windows utiliza dez mensagens para a negociação dos parâmetros de segurança. O 11° pacote, destacado no quadro superior, está protegido pelo AH (observar a linha destacada no quadro *Packet Details*), de acordo com o especificado na descrição da política de segurança.

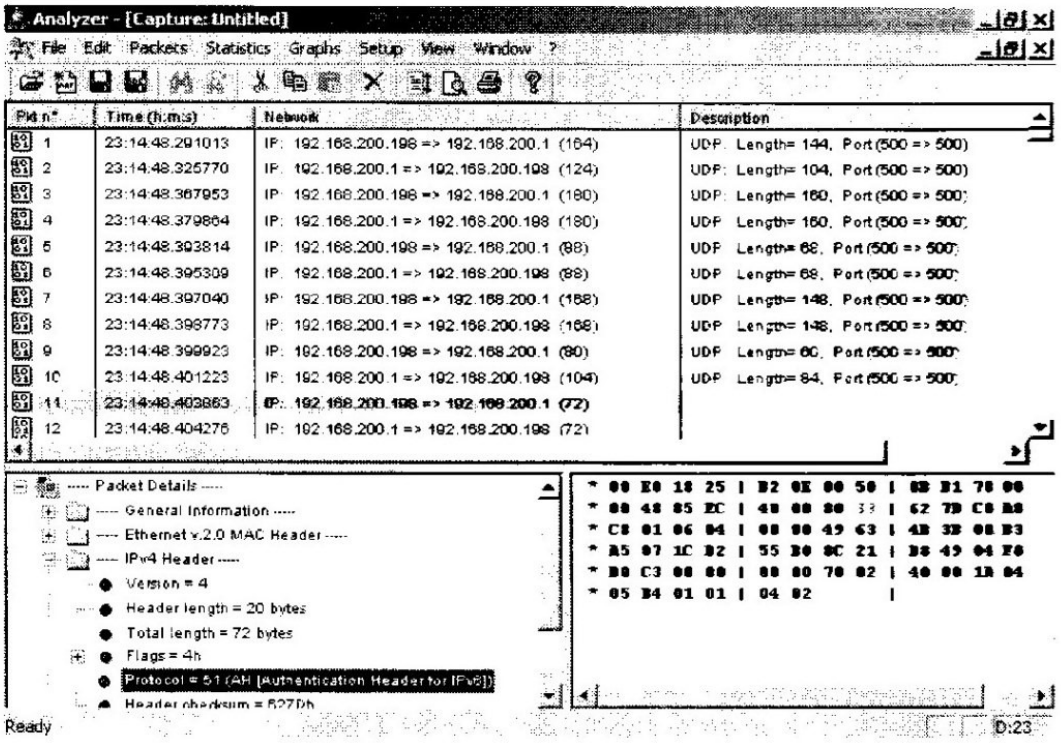

Figura 7.1: Utilização do IPsec AH

A Figura 7.2 apresenta a tela do mesmo programa analisador, com informações sobre os pacotes transmitidos quando a política de segurança especificava o uso do protocolo ESP. A partir do 11° pacote o protocolo ESP estava sendo utilizado, de forma que a política especificada na descrição de segurança foi aplicada corretamente.

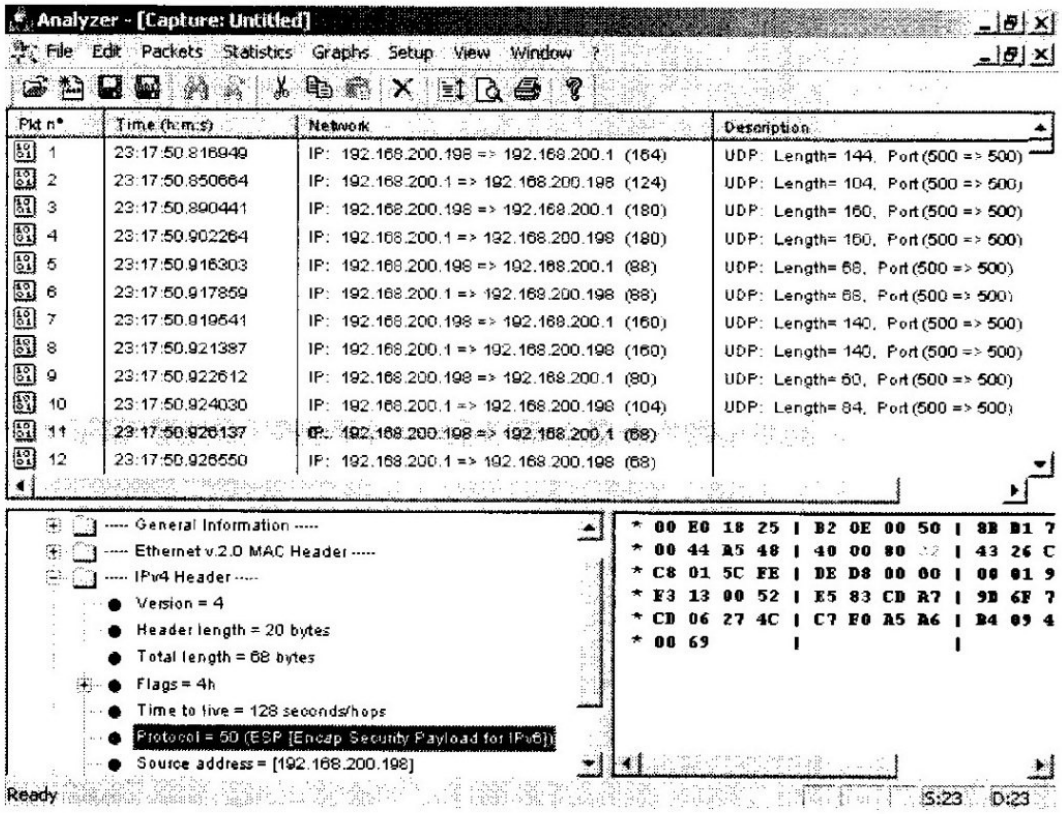

Figura 7.2: Utilização do IPsec ESP

O tamanho dos dados utilizados na chamada ao *Web Service* é configurável. Nos testes foram utilizados dados com 10, 1.000 e 10.000 bytes. Os pacotes, que trafegam pela rede, possuem um tamanho maior devido aos cabeçalhos dos protocolos SOAP e HTTP, do cabeçalhos da pilha de protocolos e do IPsec (quando utilizado). Nos testes realizados sem o IPsec, os pacotes possuíam 511 bytes a mais que os dados no caminho do cliente para o servidor (acréscimo devido aos cabeçalhos SOAP, TCP e IP), mais um pacote de 286 bytes com o cabeçalho do HTTP. Na resposta, os pacotes possuíam 531 bytes de tamanho (cabeçalhos SOAP, TCP e IP), mais um cabeçalho HTTP de 229 bytes. Com o IPsec, além dos valores já citados, ainda houve um acréscimo de até 59 bytes por pacote (os aumentos foram de 24 bytes para o protocolo AH com algoritmo MD5, de 19 a 23 bytes para o ESP com DES, de 23 a 27 bytes para o ESP com MD5 e de 31 a 35 bytes para o ESP com DES e MD5. Se o AH for utilizado junto com o ESP os acréscimos são somados).

A Tabela 7.1 apresenta as médias de tempo obtidas para 10.000 chamadas com dados de tamanho de 10 bytes. A ordenação foi feita pelo tempo médio das chamadas, desconsiderando o tempo da primeira chamada (que é muito superior que a média das

demais). A representação para os protocolos e algoritmos é a seguinte: Protocolo[Algoritmo]. A coluna Impacto é o percentual de acréscimo no tempo gasto para a realização da chamada.

Pode-se observar que os tempos de autenticação dos protocolos AH e ESP, utilizados com os algoritmos MD5 e SHA, são próximos (diferença inferior a 0,1 ms). Em relação à não utilização do IPsec, o protocolo de autenticação causou um atraso 0,5 ms, o que corresponde a 6,91%.

O impacto causado pelos algoritmos de criptografia foi um pouco maior, variando de 10,09% a 13,46%. Combinando AH com MD5 e ESP com DES, para prover autenticação e criptografia, o impacto foi de 14,59%.

Tabela 7.1: Tempos para Dados com Tamanho de 10 Bytes

	1ª chamada			**Demais chamadas**		
	Média (ms)	**Desvio Padrão**	**Impacto (%)**	**Média (ms)**	**Desvio Padrão**	**Impacto (%)**
Sem IPsec	425,11	5,30	-	7,23	2,40	-
ESP[MD5]	509,67	29,47	19,89	7,65	2,46	5,80
AH[SHA]	491,89	3,44	15,71	7,66	2,43	5,90
AH[MD5]	506,33	24,72	19,11	7,69	2,49	6,39
ESP[SHA]	495,00	7,58	16,44	7,73	2,45	6,91
ESP[DES]	502,11	24,15	18,11	7,96	2,45	10,09
ESP[3DES]	495,89	4,88	16,65	8,21	2,43	13,46
AH[MD5]+ESP[DES]	497,33	6,91	16,99	8,29	2,43	14,59

A Figura 7.3 apresenta um gráfico com os tempos médios das chamadas (desconsiderando a primeira), para dados com tamanho de 10 bytes.

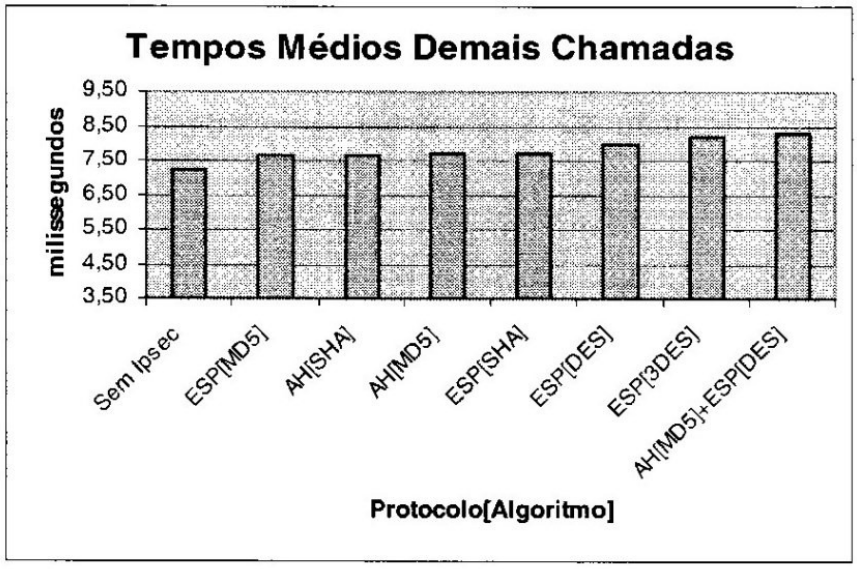

Figura 7.3: Tempos Médios Obtidos para Dados com 10 Bytes

Na Tabela 7.2 são apresentadas as médias de tempo obtidas para 10.000 chamadas com dados de tamanho de 1.000 bytes. Apesar do tempo médio ter aumentado, percentualmente o impacto não sofreu grandes alterações. Como os tempos de autenticação dos algoritmos MD5 e SHA dos protocolos ESP e AH são bem próximos, ocorreu uma inversão na ordem em que eles aparecem nesta tabela em relação à Tabela 7.1 (a diferença entre a menor e a maior média é de apenas 0,1 ms). A única grande variação foi relacionada ao algoritmo de criptografia 3DES, cujo impacto aumentou de 13,46% para 21,71%.

Tabela 7.2: Tempos para Dados com Tamanho de 1.000 Bytes

	1ª chamada			**Demais chamadas**		
	Média (ms)	**Desvio Padrão**	**Impacto (%)**	**Média (ms)**	**Desvio Padrão**	**Impacto (%)**
Sem IPsec	424,00	4,67	-	9,37	2,49	-
AH[MD5]	493,89	7,01	16,48	9,88	2,42	5,40
ESP[MD5]	497,44	16,69	17,32	9,89	2,77	5,48
AH[SHA]	500,78	8,38	18,11	9,93	2,40	5,93
ESP[SHA]	500,56	8,68	18,06	9,98	2,43	6,43
ESP[DES]	499,67	15,54	17,85	10,48	2,70	11,80
AH[MD5]+ESP[DES]	497,56	4,93	17,35	10,78	2,51	15,03
ESP[3DES]	487,44	38,35	14,96	11,41	3,60	21,71

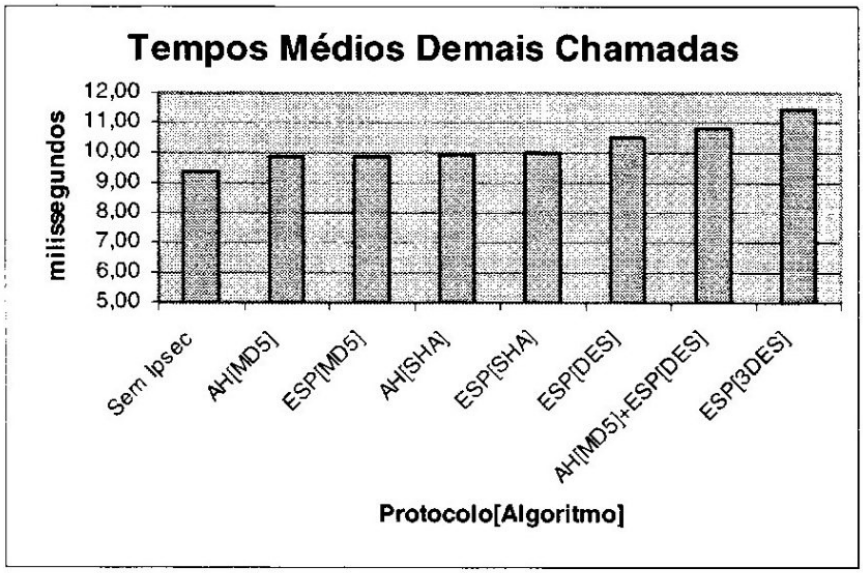

Figura 7.4: Tempos Médios Obtidos para Dados com 1.000 Bytes

A Tabela 7.3 apresenta as médias de tempo obtidas para 10.000 chamadas com dados de tamanho de 10.000 bytes. Nesse caso, a diferença de impacto entre os algoritmos avaliados foi mais significativa.

Tabela 7.3: Tempos para Dados com Tamanho de 10.000 Bytes

	1ª chamada			**Demais chamadas**		
	Média (ms)	**Desvio Padrão**	**Impacto (%)**	**Média (ms)**	**Desvio Padrão**	**Impacto (%)**
Sem IPsec	499,67	8,98	-	30,80	3,92	-
ESP[MD5]	581,89	19,22	16,46	31,46	3,66	2,16
AH[MD5]	568,78	40,40	13,83	31,82	3,80	3,32
AH[SHA]	603,11	18,64	20,70	32,22	4,02	4,61
ESP[SHA]	579,78	11,77	16,03	32,47	4,41	5,42
ESP[DES]	577,56	4,77	15,59	35,34	5,40	14,75
AH[MD5]+ESP[DES]	585,33	7,18	17,14	36,45	5,89	18,35
ESP[3DES]	590,67	15,98	18,21	38,44	5,5	24,81

É importante considerar que o tamanho de um pacote IP é limitado (no ambiente em que as avaliações foram realizadas esse limite é de 1.500 bytes), portanto, quando o pacote possui um tamanho maior que esse limite, é necessário segmentá-lo. Por isso, o acréscimo no

tamanho referente aos cabeçalhos da pilha de protocolos de rede é multiplicado pelo número de segmentos, além de serem necessários vários pacotes para transportar todos os dados.

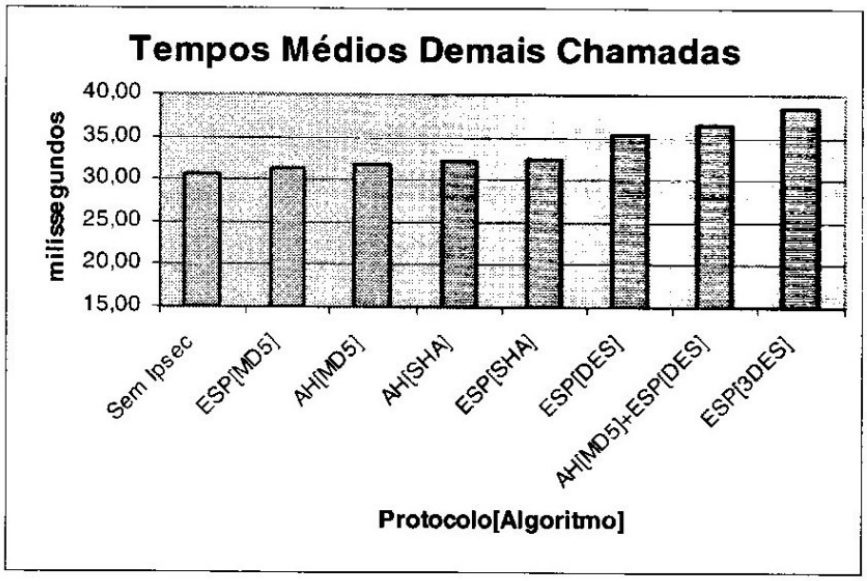

Figura 7.5: Tempos Médios Obtidos para Dados com 10.000 Bytes

A Tabela 7.4 apresenta o tempo médio gasto para a negociação das associações de segurança do IPsec. Esse tempo foi obtido com base nos valores médios sem e com a utilização do IPsec na realização da primeira chamada.

Tabela 7.4: Tempo Médio de Negociação do IPsec

Tamanho dos Dados (bytes)	Média de Tempo p/ Negociação (ms)	Desvio Padrão
10	74,63 ms	6,48
1.000	72,76 ms	4,74
10.000	84,21 ms	10,85

7.5. Conclusão do Capítulo

Com o intuito de validar a API, o esquema XML e os documentos de transformação propostos foi desenvolvido um protótipo que incluía uma aplicação cliente e um *Web Service*. O *Web Service* foi implementado da forma mais simples possível, para que a sua execução não influenciasse os resultados, e o cliente possuía os métodos necessários para realizar a mensuração do tempo gasto com as chamadas.

O MS Windows 2000 implementa apenas alguns algoritmos para os protocolos AH e ESP, portanto, não foi possível avaliar qual o impacto causado por outros algoritmos. Pelo mesmo motivo, não foram realizadas avaliações sobre o impacto causado pelo protocolo IPComp.

Os resultados obtidos comprovaram o funcionamento do mecanismo proposto nesse trabalho, e mostraram o impacto causado pelo uso do IPsec, que nas avaliações variou de 2,16% a 24,81% para os cenários utilizados.

O impacto causado pelo uso de autenticação não sofreu grandes variações com a variação do tamanho dos dados. Por outro lado, o impacto da criptografia aumentou com o crescimento do tamanho dos dados (para 10 bytes o impacto é de 14,59%, subindo para 21,71% para 1.000 bytes e atingindo 24,81% para 10.000 bytes). Dessa forma, deve-se avaliar a real necessidade de utilizar a criptografia se a quantidade de dados for grande e o desempenho for importante.

Capítulo 8

Conclusões e Trabalhos Futuros

Esse trabalho apresentou uma proposta de um mecanismo de segurança para permitir transações seguras entre clientes e *Web Services*.

Baseado no IPsec, esse mecanismo consiste de um modelo de representação de políticas em XML, uma API e um conjunto de documentos de transformação XSL, que fazem a extração de requisitos mínimos e a conversão das políticas em comandos para o dispositivo IPsec.

Quando esse trabalho foi desenvolvido, o IETF estava trabalhando em um modelo de classes para representação de políticas IPsec [JAS02], sendo o mesmo utilizado como ponto de partida para a definição da estrutura do modelo proposto. As classes foram mapeadas para XML seguindo a nomenclatura definida pelo IETF.

Para integrar esse modelo com os *Web Services* foi criada uma extensão para o documento WSDL, de forma que ele contenha as informações sobre a política de segurança junto com a descrição das interfaces de comunicação. Dessa forma, quando um cliente desejar implementar uma aplicação para realizar chamadas aos *Web Services*, ele pode obter todas as informações necessárias em um único arquivo. Dessa maneira, a obtenção de informações para acesso aos serviços é centralizada e simplificada.

O modelo de representação de políticas apresentado pode ser utilizado com propósitos diferentes de aplicações *Web Services*. Por exemplo, pode ser utilizado por um administrador para representar e configurar todos os dispositivos IPsec existentes em uma rede interna, ou ainda, pelo próprio dispositivo para armazenar localmente as informações sobre as políticas.

Os resultados, obtidos com os testes realizados no protótipo do estudo de caso, mostraram que o modelo criado é capaz de representar corretamente as políticas IPsec. Também comprovaram o funcionamento da API e dos documentos de transformação, necessários para aplicação das políticas.

Fica a cargo de trabalhos futuros criarem representações para outros mecanismos de proteção, entre eles o SSL, o XML *Signature* e o XML *Encryption*, e também uma aplicação para construção automática dos documentos de requisitos mínimos e de conversão da política para os comandos do dispositivo IPsec. Outro trabalho futuro é um mecanismo a ser implementado pelo servidor para diferenciar os clientes e dessa forma fornecer um documento WSDL específico para cada um (assunto explicado com detalhes na página 96).

Referências Bibliográficas

[ADL01] Adler, S., Berglund, A., Caruso, J., Deach, S., Graham, T., Grosso, P., Gutentag, E., Milowski, A., Parnell, S., Richman, J., Zilles, S. *Extensible Stylesheet Language (XSL) Versão 1.0*. W3C Recommendation, http://www.w3.org/TR/xsl, Outubro 2001.

[AND96] Anderson, R., Biham, E., *Fast Software Encryption*. Springer LNCS v. 1039, 1996.

[ATK02] Atkinson, B., Della-Libera, G., Hada, S., Hondo, M., Hallam-Baker, P., Klein, J., LaMacchia, B., Leach, P., Manferdelli, J., Maruyama, H., Nadalin, A., Nagaratnam, N., Prafullchandra, H., Shewchuk, J., Simon, D. *Web Services Security (WS-Security) Version 1.0*, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/. Abril 2002.

[BAR01] Bartel, M., Boyer, J., Fox, B., LaMacchia, B., Simon, E. *XML-Signature Syntax and Processing*, W3C Proposed Recommendation, http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/, Agosto 2001.

[BEL02] Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y. L., Januszewski, K., Lee, S., McKee, B., Munter, J., von Riegen, C. *UDDI Version 3.0*. Published Specification, http://uddi.org/pubs/uddi-v3.00-published-20020719.htm, Julho 2002.

[BER00] Bergholz, A. *Extending Your Markup: An XML Tutorial*. Internet Computing, Vol: 4, No. 4, Julho/Agosto 2000, p: 74-79.

[BER96] Berners-Lee, T., Fielding, R., Frystyk, H. *Hypertext Transfer Protocol – HTTP/1.0*. Internet RFC 1945. Maio 1996.

[BER98] Berners-Lee, T., Fielding, R., Masinter, L. *Uniform Resource Identifiers (URI): Generic Syntax*. Internet RFC 2396, Agosto 1998.

[BIR01] Biron, P. V., Malhotra, A. *XML Schema Part 2: Datatypes*. W3C Recommendation, http://www.w3.org/TR/xmlschema-2/, Maio 2001.

[BOX00] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., Winer, D. *Simple Object Access Protocol (SOAP) 1.1.* W3C Note, http://www.w3.org/TR/SOAP/, Maio 2000.

[BRA00] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. *Extensible Markup Language (XML) 1.0 (Second Edition).* W3C Recommendation, http://www.w3.org/TR/REC-xml, Outubro 2000.

[CHE01] Chester, T.M. *Cross-platform integration with XML and SOAP.* IT Professional , Volume: 3 Issue: 5 , Sep/Oct 2001, p: 26 -34.

[CHR01] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S. *Web Services Description Language (WSDL) 1.1.* W3C Note, http://www.w3.org/TR/wsdl, Março 2001.

[CIS03] *An Introduction to IP Security (IPSec) Encryption.* Cisco Documentation, http://www.cisco.com/warp/public/105/IPSECpart1.html, Maio 2003.

[CLA99] Clark, J. DeRose, S. *XML Path Language (XPath) Version 1.0.* W3C Recommendation, http://www.w3.org/TR/xpath, Novembro 1999.

[DAW98] Dawson, F., Stenerson, D.. *Internet Calendaring and Scheduling Core Object Specification (iCalendar).* Internet RFC 2445, Novembro 1998.

[DEE98] Deering, S., Hinden, R. *Internet Protocol, Version 6 (IPv6) Specification.* Internet RFC 2460, Dezembro 1998.

[DOW98] Dowd, P.W.; McHenry, J.T. *Network security: it's time to take it seriously.* Computer, Volume: 31 Issue: 9 , Sep 1998, p: 24 -28.

[ELK02] Elkeelany, O., Matalgah, M. M., Sheikh, K. P., Thaker, M., Chaudhry, G., Medhi, D., Qaddour, J. *Performance Analysis of IPSec Protocol: Encryption and Authentication.* IEEE Communications, 2002. ICC 2002. IEEE International Conference on, Volume 2, 2002. Page(s): 1164-1168 vol 2.

[FAL01] Fallside, D. C. *XML Schema Part 0: Primer.* W3C Recommendation, http://www.w3.org/TR/xmlschema-0/, Maio 2001.

[FER00] Ferguson, N., Schneier, B. *A Cryptographic Evaluation of IPsec.* CounterPane, Janeiro 2000.

[FER02] Fernandez, E. *Web Services Security – Current status and future.* Disponível em http://www.webservicesarchitect.com/content/articles/fernandez01.asp. Março de 2002.

[FRI98] Friend, R., and R. Monsour. *IP Payload Compression Using LZS*. Internet RFC 2395, Agosto 1998.

[FUR02a] Furst, K.; Schmidt, T.; Wippel, G.; Managing access in extended enterprise networks. Internet Computing, IEEE , Volume: 6 Issue: 5 , Setembro-Outubro de 2002 . Page(s): 67 -74.

[FUR02b] Fürst, K., Schmidt, T. Wippel, G. *Flexible Low-Cost Internet Extended-Enterprise (FloCI-EE)*. IEEE Internet Computing, Outubro 2002.

[GLE98] Glenn, R., Kent, S. *The NULL Encryption Algorithm and Its Use With IPsec*. Internet RFC 2410. Novembro 1998.

[HAD00] Hada, S., Kudo, M.. *XML Access Control Language: Provisional Authorization for XML Documents*. http://www.trl.ibm.com/projects/xml/xss4j/docs/xacl-spec.html, Outubro 2000.

[HAR02] Harkins, D., Kent, S., Perlman, R. *Proposal for the IKEv2 Protocol*. Internet draft, draft-ietf-ipsec-ikev2-01.txt. Fevereiro 2002.

[HAR98] Harkins, D., Carrel, D. *The Internet Key Exchange (IKE)*. Internet RFC 2409. Novembro 1998.

[IMA02] Imamura, T., Dillaway, B., Simon , E. *XML Encryption Syntax and Processing*. W3C Recommendation, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/. Dezembro 2002.

[JAS02] Jason, J., Rafalow, L., Vyncke, E. *IPsec Configuration Policy Model*. Internet draft draft-ietf-ipsp-config-policy-model-06.txt. Agosto 2002.

[JEP01] Jepsen, T. *SOAP Cleans up Interoperability Problems on the Web*. IT Professional, Vol. 3, No. 1, January/February 2001, p. 52-55.

[KAU01] Kaufman, C., Perlman, R., Krywaniuk, A. *Code-preserving Simplifications and Improvements to IKE*. Internet draft, draft-kaufman-ipsec-improveike-00.txt, Julho 2001.

[KEN91] Kent, S. *Security Options for the Internet Protocol*. Internet RFC 1108, Novembro 1991.

[KEN98a] Kent, S., Atkinson, R. *Security Architecture for the Internet Protocol*. Internet RFC 2401, Novembro 1998.

[KEN98b] Kent, S., Atkinson, R. *IP Authentication Header (AH)*. Internet RFC 2402, Novembro 1998.

[KEN98c] Kent, S., Atkinson, R. *IP Encapsulating Security Payload (ESP).* Internet RFC 2406. Novembro 1998.

[KER97] Keromytis, A.D. Ioannidis, J.; Smith, J.M. *Implementing IPsec.* Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE , Vol: 3, 3-8 Nov 1997, p: 1948 -1952.

[KLE01] Klensin, J. *Simple Mail Transfer Protocol.* Internet RFC 1123. Abril 2003.

[KOH93] Kohl, J., Neuman, C. *The Kerberos Network Authentication Service (V5).* Internet RFC 1510. Setembro 1993.

[MAD98a] Madson, C.. Glenn, R. *The Use of HMAC-MD5-96 within ESP and AH.* Internet RFC 2403. Novembro 1998.

[MAD98b] Madson, C.. Glenn, R. *The Use of HMAC-SHA-1-96 within ESP and AH.* Internet RFC 2404. Novembro 1998.

[MAD98c] Madson, C., Doraswamy, N. *The ESP DES-CBC Cipher Algorithm With Explicit IV.* Internet RFC 2405, Novembro 1998.

[MAL03] Maler, E. *The Future of Web Services Security: A Conversation with Eve Maler.* Disponível em http://java.sun.com/features/2003/03/webservices-qa.html. Março de 2003.

[MAU98] Maughan, D., Schertler, M., Schneider, M., Turner, J. *Internet Security Association and Key Management Protocol (ISAKMP).* Internet RFC 2408. November 1998

[MCG00] McGregor, John P., Lee, Ruby B. *Performance Impact of Data Compression on Virtual Private Network Transtactions.* Local Computer Networks, 2000. LCN 2000. Proceedings. 25th Annual IEEE Conference on, 2000. Page(s): 500-510.

[MIY00] Miyazawa, T.; Kushida, T. *An advanced Internet XML/EDI model based on secure XML documents.* Parallel and Distributed Systems: Workshops, Seventh International Conference on, 2000 , Oct 2000, p: 295 -300.

[MOO01] Moore, B. e E. Elleson, J. Strassner. *Policy Core Information Model – Version 1 Specification*, RFC 3060, Fevereiro/2001.

[MOO03] [PCIMe] Moore, B. Policy Core Information Model (PCIM) Extensions. Internet RFC 3460, Janeiro/2003.

[MUR99] Murhammer, M. W., Atakan, O., Badri, Z., Cho, B., Lee, H. J., Schmid, A. *A Comprehensive Guide to Virtual Private Networks, Volume III: Cross-Platform Key and Policy Management.* IBM Red Books, Novembro 1999.

[NAK02] Nakamur, Y.; Hada, S.; Neyama, R. *Towards the integration of Web services security on enterprise environments.* Applications and the Internet (SAINT) Workshops, 2002. Proceedings. 2002 - Symposium on , 2002, p: 166 -175.

[NIE01] Nielsen, H., Thatte, S. *Web Services Routing Protocol*, Microsoft, Outubro 2001.

[OPP98] Oppliger, R. *Security at the Internet layer.* Computer , Volume: 31 Issue: 9, Sep 1998, p: 43 -47.

[ORM98] Orman, H. *The OAKLEY Key Determination Protocol.* Internet RFC 2412. November 1998.

[PER01] Perlman, R.; Kaufman, C. *Analysis of the IPSec key exchange standard.* Enabling Technologies: Infrastructure for Collaborative Enterprises, 2001. WET ICE 2001. Proceedings. Tenth IEEE International Workshops on. Junho 2001. Pág: 150-156.

[PER98a] Pereira, R., *IP Payload Compression Using DEFLATE.* Internet RFC 2394, Agosto 1998.

[PER98b] Pereira, R., Adams, R. *The ESP CBC-Mode Cipher Algorithms.* Internet RFC 2451. Novembro 1998.

[PIP98] Piper, D. *The Internet IP Security Domain of Interpretation for ISAKMP.* Internet RFC 2407. Novembro 1998.

[RES99] Rescorla, E. *Diffie-Hellman Key Agreement Method.* Internet RFC 2631, Junho 1999.

[ROY01] Roy, J.; Ramanujan, A. *Understanding Web* services. IT Professional , Vol: 3 Issue: 6 , Nov/Dec 2001, p: 69 -73.

[SHA01] Shacham, A.; Monsour, B.; Pereira, R.; Thomas, M.. *IP Payload Compression Protocol (IPComp).* Internet RFC 3173. Setembro 2001.

[SKO00] Skonnard, A. *SOAP: The Simple Object Access Protocol.* Microsoft Internet Developer. Disponível em http://www.microsoft.com/mind/0100/soap/soap.asp. Janeiro 2000.

[TAK02] Takase, T.; Uramoto, N.; Baba, K. *XML digital signature system independent of existing applications.* Applications and the Internet (SAINT) Workshops, 2002. Proceedings, 2002, Symposium on , 2002. p: 150 -157.

[THO01] Thompson, H. S., Beech, D., Maloney, M., Mendelsohn, N. *XML Schema Part 1: Structures*. W3C Recommendation, http://www.w3.org/TR/xmlschema-1/, Maio 2001.

[TSC98] TimeStep Corporation. *Understanding the IPSec Protocol Suite*. Dezembro 1998.

[UDD01] UDDI.org. *UDDI Executive White Paper*. Disponível em www.uddi.org. Novembro 2001.

[UML03] OMG.org. *OMG Unified Modeling Language Specification*. Disponível em http://www.omg.org/docs/formal/03-03-01.pdf. Março 2003.

[WES00] Westerinen, A., Strassner, J. *CIM Core Model White Paper: Common Information Model (CIM) Core Model, Version 2.4*. DSP111. Distributed Management Task Force, Agosto 2000.

[XAV01] Xavier, E. *XML based security for e-commerce applications*. Engineering of Computer Based Systems, 2001. ECBS 2001. Proceedings. Eighth Annual IEEE International Conference and Workshop on the , 2001, p: 10 -17.

[YOU01] Younglove, R.W. *Public key infrastructure*. How it works. Computing & Control Engineering Journal , Volume: 12 Issue: 2 , Apr 2001, p: 99 -102.

ANEXO A – Descrição dos Atributos de Classes

O elemento IPsecPolicyGroup possui os seguintes atributos:

- PolicyGroupName: Nome da política;
- PolicyDecisionStrategy: Método de avaliação utilizado para as sub-políticas contidas na política. Os valores possíveis são: 1 – somente a primeira política encontrada cujas condições foram satisfeitas deve ser considerada, e 2 – todas as políticas devem ser avaliadas e todas aquelas cujas condições forem satisfeitas devem ser executadas. Se for omitido, será assumido o valor 1 (primeira política satisfeita).
- Priority: Quanto maior for o valor deste atributo maior será a prioridade de avaliação da política. Deve ser um valor único dentro do mesmo conjunto de políticas. Junto com o atributo PolicyDecisionStrategy determina a estratégia de processamento das políticas (ver explicação abaixo). Se for omitido, será assumido o valor 0 (menor prioridade).
- PolicyRoles: é uma abstração normalmente utilizada para identificar a interface dos dispositivos onde se deseja aplicar a política.

Os elementos IKERule e IPsecRule possui os seguintes atributos em comum:

- PolicyRuleName: Nome atribuído à regra.
- Enable: Indica se a regra está ativa. O valor 1 indica que a regra está ativa e o valor 2 indica que não está ativa. Se for omitido, será assumido o valor 1 (ativa).
- ConditionListType: indica se a lista de condições associadas a essa regra deve ser tratada na forma normal disjuntiva (DNF) ou na forma normal conjuntiva (CNF). O valor 1 indica DNF e o valor 2 indica CNF. Se for omitido, será assumido o valor 1 (DNF). Mais detalhes na seção 4.3.
- RuleUsage: É um texto livre que contém recomendações de como a regra deve ser usada.

- Mandatory: Indica se a avaliação da regra é obrigatória ou não. O conceito é similar à habilidade de descartar pacotes, baseado no tráfego de rede ou carga do processador. Os valores possíveis são true, indicando que a regra deve ser avaliada, e false, indicando que não é necessário avaliar a regra. Se for omitido, será assumido o valor true (a regra é obrigatória).
- SequencedActions: Indica se a seqüência de ações deve ser seguida ou não. O valor 1 indica que a seqüência deve ser seguida (seqüência obrigatória), o valor 2 indica que a seqüência é recomendada e o valor 3 indica que não importa a seqüência. Se for omitido, será assumido o valor 3 (não importa a seqüência).
- ExecutionStrategy: Determina o comportamento das ações contidas na regra. Os valores possíveis são: 1 – Executar todas as ações, 2 – executar até que uma ação seja concluída com sucesso e 3 – executar até que uma ação falhe. Exemplos de utilização: se for necessário criar um canal no modo túnel até um *gateway* e depois um canal em modo transporte até o computador destino, deve-se definir as duas ações e selecionar a opção 1 (executar todas as ações). Outro exemplo é criar duas regras, uma que define um canal a ser criado e outra que permite o tráfego sem segurança. Seleciona-se então a opção 2 (executar até que uma ação tenha sucesso) para indicar que deve ser tentado criar o canal, mas se não for possível pode-se realizar a troca de mensagens sem segurança.
- PolicyRoles: é uma abstração normalmente utilizada para identificar a interface dos dispositivos onde se deseja aplicar a política. O conteúdo é um texto livre.
- PolicyDecisionStrategy: Método de avaliação utilizado para as sub-políticas contidas na regra. Os valores possíveis são: 1 – a primeira política encontrada cujas condições foram satisfeitas deve ser considerada, e 2 – todas as políticas devem ser avaliadas e todas aquelas cujas condições forem satisfeitas devem ser executadas. Se for omitido, será assumido o valor 1 (primeira política satisfeita).
- Priority: Quanto maior for o valor deste atributo maior será a prioridade de avaliação da regra. Deve ser um valor único dentro do mesmo conjunto de

regras. Junto com o atributo PolicyDecisionStrategy determina a estratégia de processamento das regras (ver descrição na seção 6.3.2). Se for omitido, será assumido o valor 0 (menor prioridade).

- LimitNegotiation: Indica a função que o dispositivo deve ter quando for avaliar a regra. O valor 1 indica que o dispositivo deve estar iniciando a negociação, o valor 2 indica que o dispositivo deve estar respondendo a negociação (que outro dispositivo iniciou) e o valor 3 indica que o dispositivo pode assumir qualquer função.

Além desses atributos, a classe IKERule possui ainda o seguinte atributo:

- IdentityContexts: Especifica o contexto para seleção da identidade IKE a ser utilizada durante a execução da IKEAction.

O elemento SACondition possui os seguintes atributos:

- PolicyConditionName: Nome atribuído à condição.
- GroupNumber: É um número inteiro que identifica o grupo ao qual a condição pertence. O atributo ConditionListType do elemento SARule indica como deve ser o processamento. Por exemplo, se o atributo ConditionListType for DNF, então condições do mesmo grupo são avaliadas com o operador lógico AND e condições de grupos diferentes são avaliadas com o operador lógico OR (ver seção 4.3).
- ConditionNegated: Indica que o resultado da filtragem deve ser negado, ou seja, caso a condição seja avaliada para verdadeiro, o resultado será falso, e vice-versa.

O elemento FilterList representa um agrupamento de filtros, e tem somente o atributo:

- Direction: Representa a direção do fluxo de dados ao qual a lista de filtros deve ser aplicada. Os valores possíveis são: 0 – não se aplica, 1 – tráfego de entrada, 2 – tráfego de saída, 3 – ambas as direções e 4 – espelhado.

Os elementos IPSOFilterEntry, IPPeerIDPayloadFilterEntry, CredentialFilterEntry e IPHeadersFilter possuem os seguintes atributos em comum:

- Name: Nome atribuído ao filtro.

- IsNegated: Indica que o resultado da filtragem deve ser negado, ou seja, caso o filtro seja avaliado para verdadeiro, o resultado será falso, e vice-versa.

Além desses, cada filtro possui seus atributos. O elemento IPSOFilterEntry possui também os atributos:

- MatchConditionType: Especifica o campo do cabeçalho IPSO que será filtrado. Os valores possíveis são: 1 – ClassificationLevel, 2 – ProtectionAuthority.
- MatchConditionValue: Especifica o valor que o campo do cabeçalho IPSO deve conter. Alguns valores possíveis são: (A) para MatchConditionType = 1, 61 – TopSecret, 90 – Secret, 150 – Confidential e 171 – Unclassified; (B) para MatchConditionType = 2, 0 – GENSER, 1 – SIOP-ESI, 2 – SCI, 3 – NSA e 4 – DOE.

O elemento PeerIDPayloadFilterEntry possui também:

- MatchConditionType: Especifica o tipo de identidade provida pelo dispositivo IPsec no campo ID Payload. Os valores possíveis são: 1 – IPV4-ADDR, 2 – FQDN, 3 – USER_FQDN, 4 – IPV4_ADDR_SUBNET, 5 – IPV6_ADDR, 6 – IPV6_ADDR_SUBNET, 7 – IPV4_ADDR_RANGE, 8 – IPV6_ADDR_RANGE, 9 – DER_ASN1_DN, 10 – DER_ASN1_GN, 11 – KEY_ID.
- MatchConditionValue: Especifica o valor que a identificação deve conter. Esse atributo é um conjunto de caracteres. Exemplo de valores possíveis: "*@company.com", "cn=*,ou=eng,o=company,c=us" e "193.190.125.0/24".

O elemento CredentialFilterEntry possui também:

- MatchFieldName: Especifica qual parte da credencial deverá ser filtrada. Exemplo de valores possíveis: "serialNumber", "signatureAlgorithm" e "subjectName".
- MatchFieldValue: Especifica o valor que a parte da credencial indicada por MatchFieldName deve conter.
- CredentialType: indica o tipo de credencial que será utilizado. Os valores possíveis são: 1 – Certificados X.509, 2 – Tickets Kerberos.

O elemento IPHeadersFilter possui os seguintes atributos:

- HdrIpVersion: É um inteiro identificando a versão dos endereços IP a serem filtrados. Os valores possíveis são: 4 – IPv4 e 6 – IPv6. Se for omitido, então o filtro não considera a versão do endereço, e conseqüentemente os atributos HdrSrcAddress, HdrSrcAddressEndOfRange, HdrSrcMask, HdrDestAddress, HdrDestAddressEndOfRange e HdrDestMask não devem ser informados.
- HdrSrcAddress: Identifica o endereço IP de origem. Quando o atributo HdrSrcAddressEndOfRange não for fornecido, esse valor é comparado com o endereço IP origem do cabeçalho do pacote, sujeito à máscara representada pelo atributo HdrSrcMask. Quando o atributo HdrSrcAddressEndOfRange for fornecido, então o valor do atributo HdrSrcAddress é o início da faixa de endereços IP. Se for omitido, o filtro não considera o endereço de origem dos pacotes, ou seja, pacotes de qualquer origem são aceitos.
- HdrSrcAddressEndOfRange: Representa o fim da faixa de endereços IP aceitos, sendo que o início é representado pelo atributo HdrSrcAddress. Se o atributo HdrSrcAddress não for fornecido, então esse atributo também não deve ser fornecido. Se o atributo HdrSrcAddressEndOfRange for fornecido, então o atributo HdrSrcMask não deve ser fornecido.
- HdrSrcMask: Representa a máscara a ser utilizada quando for comparar os endereços de origem dos pacotes IP com o valor representado pelo atributo HdrSrcAddress. Se este atributo não for fornecido, então os endereços IP devem ser exatamente o endereço indicado pelo atributo HdrSrcAddress, ou então pertencer à faixa delimitada por HdrSrcAddress e HdrSrcAddressEndOfRange. Se o valor dessa propriedade for fornecido, então não deve ser fornecido o valor para a propriedade HdrSrcAddressEndOfRange.
- HdrDestAddress: Identifica o endereço IP de destino. Quando o atributo HdrDestAddressEndOfRange não for fornecido, esse valor é comparado com o endereço IP destino do cabeçalho do pacote, sujeito à máscara representada pelo atributo HdrDestMask. Quando o atributo

HdrDestAddressEndOfRange for fornecido, então o valor do atributo HdrDestAddress é o início da faixa de endereços IP. Se for omitido, o filtro não considera o endereço de origem dos pacotes, ou seja, pacotes de qualquer destino são aceitos.

- HdrDestAddressEndOfRange: Representa o fim da faixa de endereços IP aceitos, sendo que o início é representado pelo atributo HdrDestAddress. Se o atributo HdrDestAddress não for fornecido, então esse atributo também não deve ser fornecido. Se o atributo HdrDestAddressEndOfRange for fornecido, então o atributo HdrDestMask não deve ser fornecido.
- HdrDestMask: Representa a máscara a ser utilizada quando for comparar os endereços de origem dos pacotes IP com o valor representado pelo atributo HdrDestAddress. Se este atributo não for fornecido, então os endereços IP devem ser exatamente o endereço indicado pelo atributo HdrDestAddress, ou então pertencer à faixa delimitada por HdrDestAddress e HdrDestAddressEndOfRange. Se o valor dessa propriedade for fornecido, então não deve ser fornecido o valor para a propriedade HdrDestAddressEndOfRange.
- HdrProtocolID: É um número inteiro, representando o tipo do protocolo IP. Esse valor será comparado com o campo Protocol do cabeçalho do pacote IP. Se for omitido, todos os protocolos serão aceitos.
- HdrSrcPortStart: É um inteiro representando o limite inferior da faixa de portas UDP ou TCP de origem (inclusive). O limite superior é representado pelo atributo HdrSrcPortEnd. Se os dois atributos forem iguais, uma única porta é indicada. Se for omitido, não há limite inferior para o número de porta de origem.
- HdrSrcPortEnd: É um inteiro representando o limite superior da faixa de portas UDP ou TCP de origem (inclusive). O limite inferior é representado pelo atributo HdrSrcPortStart. Se os dois atributos forem iguais, uma única porta é indicada. Se for omitido, não há limite superior para o número de porta origem.

- HdrDestPortStart: É um inteiro representando o limite inferior da faixa de portas UDP ou TCP de destino (inclusive). O limite superior é representado pelo atributo HdrDestPortEnd. Se os dois atributos forem iguais, uma única porta é indicada. Se for omitido, não há limite inferior para o número de porta de destino.
- HdrDestPortEnd: É um inteiro representando o limite superior da faixa de portas UDP ou TCP de destino (inclusive). O limite inferior é representado pelo atributo HdrDestPortStart. Se os dois atributos forem iguais, uma única porta é indicada. Se for omitido, não há limite superior para o número de porta destino.
- HdrDSCP: Serve para filtrar o campo DSCP do cabeçalho IP (*Differentiated Services Code Point*). Usualmente possui o valor 0, mas pode ser utilizado para indicar um tipo particular de qualidade de serviço. É um conjunto de inteiros, restritos à faixa de 0 a 63.
- HdrFlowLabel: O campo *Flow Label* do cabeçalho IPv6 pode ser utilizado para rotular seqüências de pacotes para os quais é necessário um tratamento especial pelos dispositivos IPv6, como por exemplo qualidade de serviço não convencional ou serviços de tempo real. Se for omitido, o filtro não considera o campo *Flow Label* do cabeçalho IPv6.

Os atributos da classe PolicyTimePeriodCondition são:

- PolicyConditionName: Nome da condição.
- ConditionNegated: Indica que o resultado da filtragem deve ser negado, ou seja, caso a condição seja avaliada para verdadeiro, o resultado será falso, e vice-versa.
- TimePeriod: Define o período de tempo no qual a regra está ativa. É um conjunto de caracteres formatado de acordo com a RFC 2445 [DAW98] representando a data e hora de início e a data e hora de fim. O caracter 'T' separa a data da hora, e o caracter '/' separa o início do fim. Exemplo: '20000101T080000/20010131T120000' representa o início em 1º de janeiro de 2000, 8 horas da manhã e fim em 31 de janeiro de 2001 ao meio dia. A data e hora de inicio podem ser substituídas por THISANDPRIOR,

indicando que a regra já é válida e perde validade na data e hora indicada após a barra (/). A data e hora de fim podem ser substituídas por THISANDFUTURE, indicando que a regra estará sempre ativa após a data e hora de ativação.

- MonthOfYearMask: Esse atributo é usado para refinar a definição de período de validade indicado pelo atributo TimePeriod. Esse atributo indica quais os meses do ano em que a regra está ativa. Essa propriedade é formatada como um conjunto de dois caracteres, consistindo de 12 bits identificando os 12 meses do ano, seguidos de 4 bits que tem sempre o valor zero. Para os meses em que a regra estiver ativa, os bits correspondentes deverão estar marcados como 1. Os demais bits deverão conter o valor 0.
- DayOfMonthMask: Tem o mesmo objetivo do atributo MonthOfYearMask, mas serve para indicar os dias do mês nos quais a regra está ativa. Essa propriedade é formatada como um conjunto de 8 caracteres, consistindo de 31 bits identificando os dias do mês contando a partir do início do mês, seguidos de 31 bits indicando os dias do mês contando a partir do final do mês, mais 2 bits sempre com valor 0.
- DayOfWeekMask: Semelhante ao atributo DayOfMonthMask, mas indica os dias da semana nos quais a regra está ativa. Esse atributo é formatado como 1 caracter, sendo que 7 bits indicam os dias da semana começando no domingo e o último bit contém sempre o valor 0.
- TimeOfDayMask: Esse atributo é usado para refinar a definição de período de validade indicado pelo atributo TimePeriod. Esse atributo indica as horas do dia em que a regra está ativa. Esse atributo é formatado como definido na RFC 2445, o caracter 'T' seguido da hora de início, seguido do caracter separador '/' e o caracter 'T' seguido da hora de fim.
- LocalOrUtcTime: Indica se a hora representada no atributo TimePeriod e nos demais atributos que envolvem hora são horário local ou horário UTC25. Os valores possíveis são: 1 indica hora local e 2 indica hora UTC. Se for omitido, é assumido o valor 2.

[25] Universal Time Coordinated: padrão internacional para hora atual, baseada em relógios atômicos.

•

Todos os elementos de ação contêm os seguintes atributos em comum:

- PolicyActionName: Nome da ação.
- DoActionLogging: Especifica que uma mensagem de *log* deve ser gerada sempre que a ação for executada. No caso da ação representar uma negociação, a mensagem de *log* será gerada independente do sucesso ou falha da negociação. É um valor booleano.
- DoPacketLogging: Especifica que uma mensagem de *log* deve ser gerada sempre que a associação de segurança resultante da ação for utilizada para processar um pacote. É um valor booleano.
- ActionOrder: Especifica a posição relativa da ação na seqüência de ações associadas à regra. Valores menores têm precedência sobre valores maiores.

O elemento CompoundPolicyAction possui os seguintes elementos:

- SequencedActions: Indica se a seqüência de ações deve ser seguida ou não. O valor 1 indica que a seqüência deve ser seguida, o valor 2 indica que a seqüência é recomendada e o valor 3 indica que não importa a seqüência. Se for omitido, será assumido o valor 3 (não importa a seqüência).
- ExecutionStrategy: Determina o comportamento das ações contidas na regra. Os valores possíveis são: 1 – Executar todas as ações, 2 – executar até que uma ação seja concluída com sucesso e 3 – executar até que uma ação falhe. Exemplos de utilização: se for necessário criar um canal no modo túnel até um *gateway* e depois um canal em modo transporte até o computador destino, deve-se definir as duas ações e selecionar a opção 1 (executar todas as ações). Outro exemplo é criar duas regras, uma que define um canal a ser criado e outra que permite o tráfego sem segurança. Seleciona-se então a opção 2 (executar até que uma ação tenha sucesso).

O IKEAction possui os seguintes atributos:

- ExchangeMode: Especifica qual modo o IKE deve utilizar apra a negociação da fase um. Os valores possíveis são: 1 – Base, 2 – Main e 3 – Agressive.

- UseIKEIdentityType: Especifica o tipo de identidade que deve ser utilizado durante a negociação. Os valores possíveis são: 1 – IPV4-ADDR, 2 – FQDN, 3 – USER_FQDN, 4 – IPV4_ADDR_SUBNET, 5 – IPV6_ADDR, 6 – IPV6_ADDR_SUBNET, 7 – IPV4_ADDR_RANGE, 8 – IPV6_ADDR_RANGE, 9 – DER_ASN1_DN, 10 – DER_ASN1_GN, 11 – KEY_ID.
- MinLifetimeSeconds: tempo de vida mínimo em segundos que será aceito em uma negociação. É utilizado para prevenir ataques de negação de serviço. É um número inteiro.
- MinLifetimeKilobytes: tempo de vida mínimo em KBytes que será aceito em uma negociação. É utilizado para prevenir ataques de negação de serviço. É um número inteiro.
- IdleDurationSeconds: Especifica o tempo em segundos que uma associação de segurança pode ficar ociosa (isto é, sem proteger qualquer tráfego) antes de ser apagada. É um número inteiro.
- AgressiveModeGroupId: Especifica qual *group ID* será utilizado nos primeiros pacotes da negociação da fase 1. Somente será utilizado se o atributo ExchangeMode estiver configurado para o modo Agressive. Se o atributo AgressiveModeGroupId estiver na faixa específica de fornecedores (32768 a 65535), então o atributo VendorID identifica o número do grupo (group ID). Os valores possíveis são: 1 – default 768-bit MODP group, 2 – alternate 1024-bit MODP group, 3 – EC2N group on GP[2^155] e 4 – EC2N group on GP[2^185].
- VendorID: Especifica o valor a ser utilizado no campo Vendor ID. É um conjunto de caracteres.

As classes IPsecTransportAction e IPsecTunnelAction possuem os seguintes atributos em comum:

- UsePFS: indica se deve ser utilizado o *Perfect Forward Secrecy* (a criação de novas chaves não depende das chaves utilizadas anteriormente).
- UseIKEGroup: Especifica se a fase 2 da negociação deve utilizar o mesmo *Group ID* utilizado na fase 1. Esse atributo é ignorado se o atributo UsePFS for falso. É um valor booleano.

- GroupId: Especifica o grupo de troca de chaves das associações de segurança. para a fase 2. Esse atributo é ignorado se o atributo UsePFS for falso ou se o atributo UsePFS for verdadeiro e a atributo UseIKEGroup também for verdadeiro. Se o atributo GroupId estiver na faixa específica de fornecedores (32768 a 65535), então o atributo VendorID identifica o número do grupo (group ID). Os valores possíveis são: 1 – default 768-bit MODP group, 2 – alternate 1024-bit MODP group, 3 – EC2N group on GP[2^155] e 4 – EC2N group on GP[2^185].
- Granularity: Especifica como as informações para a associação de segurança devem ser obtidas dos pacotes que iniciaram a negociação. Os valores possíveis são: 1 – subrede – as máscaras de origem e destino dos filtros serão utilizados, 2 – endereço – somente os endereços IP de origem e destino serão utilizados, 3 – protocolo – os endereços IP de origem e destino e o protocolo IP serão utilizados, 4 – porta – os endereços IP de origem e destino, o protocolo IP e as portas de origem e destino (camada 4) serão utilizados.
- MinLifetimeSeconds: tempo de vida mínimo em segundos que será aceito em uma negociação. É um número inteiro.
- MinLifetimeKilobytes: tempo de vida mínimo em KBytes que será aceito em uma negociação. É um número inteiro.
- IdleDurationSeconds: Especifica o tempo em segundos que uma associação de segurança pode ficar ociosa (isto é, sem proteger qualquer tráfego) antes de ser apagada. É um número inteiro.
- VendorID: É usado junto com o atributo GroupID (quando este for da faixa específica de fornecedores) para identificar o grupo para troca de chaves. Esse atributo é ignorado a menos que o atributo UsePFS seja *true* e o atributo UseIKEGroup seja *false* e o atributo GroupID seja da faixa específica de fornecedores. É um conjunto de caracteres.

Além desses atributos, o elemento IPsecTunnelAction possui ainda o atributo:

- DFHandling: Especifica como o bit *Don't Fragment* (DF) do cabeçalho IP deve ser manipulado durante o processamento do IPsec, quando for utilizado o modo túnel. Os valores possíveis são: 1 – copiar,

- o bit DF do cabeçalho IP interno para o cabeçalho externo, 2 – atribuir o valor 1 para o bit DF do cabeçalho IP externo, 3 – atribuir o valor 0 para o bit DF do cabeçalho externo.

O elemento PeerGateway (contido no elemento IPsecTunnelAction) possui os seguintes atributos:

- Name: Nome do computador ou *firewall* até o qual será criado o túnel IPsec.
- PeerIdentityType: Especifica o tipo de identificação IKE utilizado. Os valores possíveis são: 1 – IPV4-ADDR, 2 – FQDN, 3 – USER_FQDN, 4 – IPV4_ADDR_SUBNET, 5 – IPV6_ADDR, 6 – IPV6_ADDR_SUBNET, 7 – IPV4_ADDR_RANGE, 8 – IPV6_ADDR_RANGE, 9 – DER_ASN1_DN, 10 – DER_ASN1_GN, 11 – KEY_ID.
- PeerIdentity: Identificação do computador ou *firewall*. Deve ser do tipo indicado pelo atributo PeerIdentityType.
- SequenceNumber: Especifica a ordem avaliação dos elementos PeerGateway. Pode-se especificar mais de um PeerGateway a ser utilizado, de forma que se não for possível criar o túnel um, pode-se ir tentando outros PeerGateway até que seja possível criar o canal. É um número inteiro, valores menores são avaliados primeiro.

O elemento IKEProposal possui os seguintes atributos:

- Name: Define um nome para a proposta IKE.
- CipherAlgorithm: especifica o algoritmo de criptografia a ser utilizado pela fase 1 do IKE. Os valores possíveis são: 1 – DES-CBC, 2 – IDEA-CBC, 3 – Blowfish-CBC, 4 – RC5-R16-B64-CBC, 5 – 3DES-CBC e 6 – CAST-CBC.
- HashAlgorithm: especifica o algoritmo de *hash* para a fase 1 do IKE. Os valores possíveis são: 1 – MD5, 2 – SHA e 3 – Tiger [AND96].
- PRFAlgorithm: especifica a função pseudo-aleatória para a fase 1 do IKE. Atualmente não existe nenhuma função pré-definida.
- GroupId: especifica o grupo de troca de chaves das associações de segurança. Esse atributo é ignorado no modo agressivo. Os valores

possíveis são: 1 – default 768-bit MODP group, 2 – alternate 1024-bit MODP group, 3 – EC2N group on GP[2^155] e 4 – EC2N group on GP[2^185]. Os valores entre 32768 e 65535 são específicos para fabricantes.

- AuthenticationMethod: especifica o método de autenticação para a fase 1 do IKE. Os valores válidos são: 1 – chave pré-compartilhada, 2 – assinaturas DSS, 3 – assinaturas digitais RSA, 4 – criptografia de chave pública com RSA e 5 – criptografia de chave pública revisada com RSA, 6 - Kerberos. Se for utilizado o valor 0, significa que deve haver uma proposta para cada credencial existente no sistema (exemplo, se tiver uma chave pré-compartilhada e um certificado, então deve haver 2 propostas, uma para cada método). Conforme apresentado na seção 5.4, o método de chave pré-compartilhada não é indicado para aplicações na Internet. Portanto não deve ser utilizado.
- MaxLifetimeSeconds: especifica o tempo máximo em segundos que a associação de segurança proposta permanecerá válida após a sua criação. É um número inteiro não negativo. O número zero indica o valor padrão de 8 horas.
- MaxLifetimeKilobytes: especifica a quantidade máxima de dados em Kbytes que a associação de segurança proposta irá proteger antes de ser apagada. É um número inteiro não negativo. O número zero indica que não há tempo máximo.
- VendorID: também é utilizado para qualificar o grupo de troca de chaves. Esse atributo é ignorado se for utilizado o modo agressivo ou se o atributo GroupID não estiver na faixa específica para fabricantes. É um texto livre.
- SequenceNumber: especifica a ordem de preferência das propostas. É um inteiro não negativo. Valores menores têm preferência sobre valores maiores.

O elemento IPsecProposal contém os seguintes atributos:

- Name: Define um nome para a proposta IPsec.

- SequenceNumber: especifica a ordem de preferência das propostas. É um inteiro não negativo. Valores menores têm preferência sobre valores maiores.

Os elementos AHTranform, ESPTransform e IPCOMPTransform possuem alguns atributos em comum. São eles:

- CommonName: Nome atribuído à transformação.
- MaxLifetimeSeconds: Especifica o tempo máximo em segundos durante o qual a associação de segurança proposta permanecerá válida após a sua criação. É um número inteiro.
- MaxLifetimeKilobytes: especifica a quantidade máxima de dados em Kbytes que a associação de segurança proposta irá proteger antes de ser apagada.
- VendorID: Identifica o grupo de troca de chaves. Esse atributo é ignorado a menos que não seja utilizado o modo agressivo e o atributo GroupID seja da faixa específica de fornecedor.
- SequenceNumber: Especifica a ordem de preferência das transformações do mesmo tipo. É um valor inteiro.

Além dos atributos citados acima, o elemento AHTransform contém ainda:

- AHTransformID: Especifica o identificador do algoritmo de transformação para o protocolo AH. Os valores possíveis são: 2 – protoçolo MD5, 3 – protocolo SHA, 4 – protocolo DES. Os valores 0 e 1 são reservados, e os demais serão utilizados para protocolos futuros.
- UseReplayPrevention: Especifica se a prevenção contra reenvio de pacotes antigos deve ser utilizada. É um valor booleano.
- ReplayPreventionWindowSize: Especifica o tamanho em bits da janela utilizada para o mecanismo de prevenção de reenvio de pacotes antigos. Esse valor é ignorado se o atributo UseReplayPrevention for falso. É um valor inteiro, e assume-se que esse número será potência de 2 [JAS02].

Da mesma forma, o elemento ESPTransform possui também os atributos:

- IntegrityTransformId: Especifica o identificador de transformação a ser utilizado para o algoritmo de integridade do ESP. Os valores possíveis são: 0 – sem integridade, 1 – MD5, 2 – SHA-1, 3 – DES.

- CipherTransformId: Especifica o identificador de transformação a ser utilizado para o algoritmo de criptografia do ESP. Os valores possíveis são: 1 – DES_IV64, 2 – DES, 3 – 3DES, 4 – RC5, 5 – IDEA, 6 – CAST, 7 – Blowfish, 8 – 3IDEA, 9 – DES_IV32, 10 – RC4, 11 – nenhum protocolo.
- CipherKeyLength: Especifica o tamanho em bits da chave para o algoritmo de criptografia.
- CipherKeyRounds: Especifica o número de execuções do algoritmo de criptografia. Para algoritmos de criptografia que utilizam um número fixo de execuções, esse valor é ignorado.
- UseReplayPrevention: Especifica se a prevenção contra reenvio de pacotes antigos deve ser utilizada. É um valor booleano.
- ReplayPreventionWindowSize: Especifica o tamanho em bits da janela utilizada para o mecanismo de prevenção de reenvio de pacotes antigos. Esse valor é ignorado se o atributo UseReplayPrevention for falso. É um valor inteiro, e assume-se que esse número será potência de 2 [JAS02].

O elemento IPCOMPTransform também possui alguns elementos além dos citados acima:

- Algorithm: Especifica o identificador de transformação a ser utilizado para o algoritmo de compressão do IPComp. Os valores possíveis são: 1 – OUI (algoritmo específico de fornecedor, especificado pelo atributo PrivateAlgorithm), 2 – DEFLATE e 3 – LZS.
- DictionarySize: especifica o tamanho do dicionário para o algoritmo de compressão. Para algoritmos com tamanho pré-definido, esse valor é ignorado. É um valor inteiro.
- PrivateAlgorithm: Especifica um algoritmo específico de fornecedor. Esse valor é ignorado se o atributo Algorithm for diferente de 1. É um valor inteiro.

ANEXO B – Esquema XML completo

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="SecurityPolicy">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="IPsecPolicyGroup" type="IPsecPolicyGroupType"
               maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="SecurityPolicyName" type="xsd:string"/>
         <xsd:attribute name="PolicyDecisionStrategy" type="xsd:nonNegativeInteger"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:complexType name="IPsecPolicyGroupType">
      <xsd:sequence>
         <xsd:element name="IKERule" type="IKERuleType" minOccurs="0"
            maxOccurs="unbounded"/>
         <xsd:element name="IPsecRule" type="IPsecRuleType" minOccurs="0"
            maxOccurs="unbounded"/>
         <xsd:element name="IPsecPolicyGroup" type="IPsecPolicyGroupType"
            minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="PolicyGroupName" type="xsd:string"/>
      <xsd:attribute name="PolicyDecisionStrategy" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="PolicyRoles" type="xsd:string"/>
      <xsd:attribute name="Priority" type="xsd:nonNegativeInteger"/>
   </xsd:complexType>
   <xsd:complexType name="IKERuleType">
      <xsd:sequence>
         <xsd:element name="SACondition" type="SAConditionType"
            maxOccurs="unbounded"/>
         <xsd:element name="PolicyTimePeriodCondition"
            type="PolicyTimePeriodConditionType" minOccurs="0"
            maxOccurs="unbounded"/>
         <xsd:element name="IKEAction" type="IKEActionType" minOccurs="0"
            maxOccurs="unbounded"/>
         <xsd:element name="IKERejectAction" type="IKERejectActionType"
            minOccurs="0"/>
         <xsd:element name="PreconfiguredSAAction" type="PreconfiguredSAActionType"
            minOccurs="0"/>
         <xsd:element name="CompoundIKEAction" type="CompoundIKEActionType"
            minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="IKERule" type="IKERuleType" minOccurs="0"
            maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="PolicyRuleName" type="xsd:string"/>
      <xsd:attribute name="Enable" type="xsd:boolean"/>
      <xsd:attribute name="ConditionListType" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="RuleUsage" type="xsd:string"/>
      <xsd:attribute name="Mandatory" type="xsd:boolean"/>
      <xsd:attribute name="SequencedActions" type="xsd:string"/>
      <xsd:attribute name="ExecutionStrategy" type="xsd:string"/>
      <xsd:attribute name="PolicyRoles" type="xsd:string"/>
      <xsd:attribute name="PolicyDecisionStrategy" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="LimitNegotiation" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="IdentityContexts" type="xsd:string"/>
```

```
   <xsd:attribute name="Priority" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IPsecRuleType">
   <xsd:sequence>
      <xsd:element name="SACondition" type="SAConditionType"
         maxOccurs="unbounded"/>
      <xsd:element name="PolicyTimePeriodCondition"
         type="PolicyTimePeriodConditionType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="IPsecTransportAction" type="IPsecTransportActionType"
         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="IPsecTunnelAction" type="IPsecTunnelActionType"
         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="IPsecBypassAction" type="IPsecBypassActionType"
         minOccurs="0"/>
      <xsd:element name="IPsecDiscardAction" type="IPsecDiscardActionType"
         minOccurs="0"/>
      <xsd:element name="PreconfiguredSAAction" type="PreconfiguredSAActionType"
         minOccurs="0"/>
      <xsd:element name="CompoundIPsecAction" type="CompoundIPsecActionType"
         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="IPsecRule" type="IPsecRuleType" minOccurs="0"
         maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyRuleName" type="xsd:string"/>
   <xsd:attribute name="Enable" type="xsd:boolean"/>
   <xsd:attribute name="ConditionListType" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="RuleUsage" type="xsd:string"/>
   <xsd:attribute name="Mandatory" type="xsd:boolean"/>
   <xsd:attribute name="SequencedActions" type="xsd:string"/>
   <xsd:attribute name="ExecutionStrategy" type="xsd:string"/>
   <xsd:attribute name="PolicyRoles" type="xsd:string"/>
   <xsd:attribute name="PolicyDecisionStrategy" type="xsd:nonNegativeInteger" />
   <xsd:attribute name="LimitNegotiation" type="xsd:nonNegativeInteger" />
   <xsd:attribute name="Priority" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="SAConditionType">
   <xsd:sequence>
      <xsd:element name="FilterList">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="IPSOFilterEntry" minOccurs="0"
                  maxOccurs="unbounded">
                  <xsd:complexType>
                     <xsd:attribute name="Name" type="xsd:string"/>
                     <xsd:attribute name="IsNegated" type="xsd:boolean"/>
                     <xsd:attribute name="MatchConditionType"
                        type="xsd:nonNegativeInteger"/>
                     <xsd:attribute name="MatchConditionValue"
                        type="xsd:nonNegativeInteger"/>
                  </xsd:complexType>
               </xsd:element>
               <xsd:element name="PeerIDPayloadFilterEntry" minOccurs="0"
                     maxOccurs="unbounded">
                  <xsd:complexType>
                     <xsd:attribute name="Name" type="xsd:string"/>
                     <xsd:attribute name="IsNegated" type="xsd:boolean"/>
                     <xsd:attribute name="MatchConditionType"
                        type="xsd:nonNegativeInteger"/>
                     <xsd:attribute name="MatchConditionValue" type="xsd:string"/>
                  </xsd:complexType>
               </xsd:element>
               <xsd:element name="CredentialFilterEntry" minOccurs="0"
                     maxOccurs="unbounded">
                  <xsd:complexType>
```

```
            <xsd:attribute name="Name" type="xsd:string"/>
            <xsd:attribute name="IsNegated" type="xsd:boolean"/>
            <xsd:attribute name="MatchFieldName" type="xsd:string"/>
            <xsd:attribute name="MatchFieldValue" type="xsd:string"/>
            <xsd:attribute name="CredentialType"
               type="xsd:nonNegativeInteger"/>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="IPHeadersFilter" minOccurs="0"
            maxOccurs="unbounded">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="HdrFlowLabel" minOccurs="0">
                  <xsd:simpleType>
                  <xsd:list itemType="xsd:nonNegativeInteger"/>
                  </xsd:simpleType>
               </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="Name" type="xsd:string"/>
            <xsd:attribute name="IsNegated" type="xsd:boolean"/>
            <xsd:attribute name="HdrIpVersion"
               type="xsd:nonNegativeInteger"/>
            <xsd:attribute name="HdrSrcAddress" type="xsd:string"/>
            <xsd:attribute name="HdrSrcAddressEndOfRange"
               type="xsd:string"/>
            <xsd:attribute name="HdrSrcMask" type="xsd:string"/>
            <xsd:attribute name="HdrDestAddress" type="xsd:string"/>
            <xsd:attribute name="HdrDestAddressEndOfRange"
               type="xsd:string"/>
            <xsd:attribute name="HdrDestMask" type="xsd:string"/>
            <xsd:attribute name="HdrProtocolID"
               type="xsd:nonNegativeInteger"/>
            <xsd:attribute name="HdrSrcPortStart"
               type="xsd:nonNegativeInteger"/>
            <xsd:attribute name="HdrSrcPortEnd"
               type="xsd:nonNegativeInteger"/>
            <xsd:attribute name="HdrDestPortStart"
               type="xsd:nonNegativeInteger"/>
            <xsd:attribute name="HdrDestPortEnd"
               type="xsd:nonNegativeInteger"/>
            <xsd:attribute name="HdrDSCP" type="xsd:nonNegativeInteger"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="Direction" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="AcceptCredentialsFrom" minOccurs="0">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="CertificateAuthority" minOccurs="0"
               maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:attribute name="CADistinguishedName"/>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="KerberosKeyDistributionCenter" minOccurs="0"
               maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:attribute name="Name"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

```
   </xsd:sequence>
   <xsd:attribute name="PolicyConditionName" type="xsd:string"/>
   <xsd:attribute name="GroupNumber" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="ConditionNegated" type="xsd:boolean"/>
</xsd:complexType>
<xsd:complexType name="PolicyTimePeriodConditionType">
   <xsd:attribute name="PolicyConditionName" type="xsd:string"/>
   <xsd:attribute name="ConditionNegated" type="xsd:boolean"/>
   <xsd:attribute name="TimePeriod" type="xsd:string"/>
   <xsd:attribute name="MonthOfYearMask" type="xsd:string"/>
   <xsd:attribute name="DayOfMonthMask" type="xsd:string"/>
   <xsd:attribute name="DayOfWeekMask" type="xsd:string"/>
   <xsd:attribute name="TimeOfDayMask" type="xsd:string"/>
   <xsd:attribute name="LocalOrUtcTime" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="CompoundIKEActionType">
   <xsd:sequence>
      <xsd:element name="IKEAction" type="IKEActionType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="IKERejectAction" type="IKERejectActionType"
         minOccurs="0"/>
      <xsd:element name="PreconfiguredSAAction" type="PreconfiguredSAActionType"
         minOccurs="0"/>
      <xsd:element name="CompoundIKEAction" type="CompoundIKEActionType"
         minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="SequencedActions" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="ExecutionStrategy" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="CompoundIPsecActionType">
   <xsd:sequence>
      <xsd:element name="IPsecTransportAction" type="IPsecTransportActionType"
         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="IPsecTunnelAction" type="IPsecTunnelActionType"
         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="IPsecBypassAction" type="IPsecBypassActionType"
         minOccurs="0"/>
      <xsd:element name="IPsecDiscardAction" type="IPsecDiscardActionType"
         minOccurs="0"/>
      <xsd:element name="PreconfiguredSAAction" type="PreconfiguredSAActionType"
         minOccurs="0"/>
      <xsd:element name="CompoundIPsecAction" type="CompoundIPsecActionType"
         minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="SequencedActions" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="ExecutionStrategy" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IKEActionType">
   <xsd:sequence>
      <xsd:element name="IKEProposal" type="IKEProposalType" minOccurs="0"
         maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="ExchangeMode" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="UseIKEIdentityType" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="IdleDurationSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="AgressiveModeGroupId" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
```

```
</xsd:complexType>
<xsd:complexType name="IPsecTransportActionType">
   <xsd:sequence>
      <xsd:element name="IPsecProposal" type="IPsecProposalType" minOccurs="0"
         maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="UsePFS" type="xsd:boolean"/>
   <xsd:attribute name="UseIKEGroup" type="xsd:boolean"/>
   <xsd:attribute name="GroupId" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="Granularity" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="IdleDurationSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IPsecTunnelActionType">
   <xsd:sequence>
      <xsd:element name="IPsecProposal" type="IPsecProposalType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="PeerGateway" minOccurs="1" maxOccurs="unbounded">
         <xsd:complexType>
            <xsd:attribute name="Name" type="xsd:string"/>
            <xsd:attribute name="PeerIdentityType" type="xsd:nonNegativeInteger"/>
            <xsd:attribute name="PeerIdentity" type="xsd:string"/>
            <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="UsePFS" type="xsd:boolean"/>
   <xsd:attribute name="UseIKEGroup" type="xsd:boolean"/>
   <xsd:attribute name="GroupId" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="Granularity" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MinLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="IdleDurationSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="DFHandling" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IKERejectActionType">
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="LifetimeSeconds" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IPsecBypassActionType">
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="LifetimeSeconds" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IPsecDiscardActionType">
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="DoActionLogging" type="xsd:boolean"/>
   <xsd:attribute name="DoPacketLogging" type="xsd:boolean"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
```

```
   <xsd:attribute name="LifetimeSeconds" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="PreconfiguredSAActionType">
   <xsd:attribute name="PolicyActionName" type="xsd:string"/>
   <xsd:attribute name="ActionOrder" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="SPI" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IKEProposalType">
   <xsd:attribute name="Name" type="xsd:string"/>
   <xsd:attribute name="CipherAlgorithm" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="HashAlgorithm" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="PRFAlgorithm" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="GroupId" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="AuthenticationMethod" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MaxLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MaxLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="AHTransformType">
   <xsd:attribute name="CommonName"/>
   <xsd:attribute name="AHTransformID" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MaxLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MaxLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="UseReplayPrevention" type="xsd:boolean"/>
   <xsd:attribute name="ReplayPreventionWindowSize"
      type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="ESPTransformType">
   <xsd:attribute name="CommonName"/>
   <xsd:attribute name="MaxLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MaxLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="IntegrityTransformId" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="CipherTransformId" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="CipherKeyLength" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="CipherKeyRounds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="UseReplayPrevention" type="xsd:boolean"/>
   <xsd:attribute name="ReplayPreventionWindowSize"
      type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IPCOMPTransformType">
   <xsd:attribute name="CommonName"/>
   <xsd:attribute name="MaxLifetimeSeconds" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="MaxLifetimeKilobytes" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="Algorithm" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="DictionarySize" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="PrivateAlgorithm" type="xsd:nonNegativeInteger"/>
   <xsd:attribute name="VendorID" type="xsd:string"/>
   <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
<xsd:complexType name="IPsecProposalType">
   <xsd:sequence>
      <xsd:element name="AHTransform" type="AHTransformType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="ESPTransform" type="ESPTransformType" minOccurs="0"
         maxOccurs="unbounded"/>
      <xsd:element name="IPCOMPTransform" type="IPCOMPTransformType" minOccurs="0"
         maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="Name" type="xsd:string"/>
   <xsd:attribute name="SequenceNumber" type="xsd:nonNegativeInteger"/>
```

```
    </xsd:complexType>
</xsd:schema>
```

ANEXO C – Código Fonte da API e do Estudo de Caso em Java

```
package ping;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.StringWriter;

import javax.rmi.CORBA.Stub;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

import org.w3c.dom.Document;
import org.xml.sax.SAXException;
import java.io.ByteArrayInputStream;

import java.util.StringTokenizer;
import java.util.Vector;
import java.net.URL;

public class PingClient {
    public static void main(String[] args) {
        try {
        int iRepeticoes = 100; //Integer.parseInt(args[1]);
        int iTamanhoDados = 10;
        boolean bUsarIPsec = false;
        long Inicio = System.currentTimeMillis();

        SecurityManager sm = new SecurityManager("xslt/swsdl.xsl",
                "xslt/swsdlRemove.xsl", "xslt/requisitos.xsl");

        if(bUsarIPsec)
           sm.loadPolicyFile("xslt/swsdl.xml");

        if(bUsarIPsec)
           sm.applyPolicy();
        long lTempoAplicacaoPolitica = System.currentTimeMillis() - Inicio;

        Stub stub = createProxy();
        PingIF ping = (PingIF)stub;
```

```
      String sDados = new String();
      for(int i = 0; i< iTamanhoDados; i++)
         sDados += Integer.toString(i % 10);


      System.out.println("--Aguardando...");
      Thread.sleep(100);
      System.out.println("--Iniciando...");

      long lTempoChamadas [] = new long[iRepeticoes];
      long lTempoAnterior;
      String sResponse;
      for(int i = 0; i < iRepeticoes; i++)
      {
         Thread.sleep(5);
         lTempoAnterior = System.currentTimeMillis();
         if(i == 0)
         {
            sResponse = ping.sayPing(sDados);
            lTempoChamadas[i] = 10*(System.currentTimeMillis() - lTempoAnterior);
         }
         else
         {
            sResponse = ping.sayPing(sDados);
            sResponse = ping.sayPing(sDados);
            sResponse = ping.sayPing(sDados);
            sResponse = ping.sayPing(sDados);
            sResponse = ping.sayPing(sDados);

            sResponse = ping.sayPing(sDados);
            sResponse = ping.sayPing(sDados);
            sResponse = ping.sayPing(sDados);
            sResponse = ping.sayPing(sDados);
            sResponse = ping.sayPing(sDados);
            lTempoChamadas[i] = System.currentTimeMillis() - lTempoAnterior;
         }
         System.out.println(Long.toString(lTempoChamadas[i]/10));
      }

      if(bUsarIPsec)
         sm.removePolicy();

      long Fim = System.currentTimeMillis();
      System.out.println("Numero de repeticoes: " + String.valueOf(iRepeticoes) +
         " - Tempo Total: " + Long.toString(Fim-Inicio) + " milisegundos");
      System.out.println("Tempo aplicacao da politica: " +
         Long.toString(lTempoAplicacaoPolitica));
      System.out.println("Tempo primeira chamada: " +
         Long.toString(lTempoChamadas[0]/10));
      System.out.println("Média todas as chamadas: " +
         Float.toString(calculaMedia(lTempoChamadas,0,99)/10));
      System.out.println("Média chamadas a partir da segunda:" +
         Float.toString(calculaMedia(lTempoChamadas, 1, 99)/10));
   } catch (Exception ex) {
      ex.printStackTrace();
   }
}

private static Stub createProxy() {
   return (Stub)(new MyPing_Impl().getPingIFPort());
}

private static float calculaMedia(long tempos[], int inicio, int fim)
{
   long lValorTotal = 0;
```

```
      for(int i = inicio; i <= fim; i++)
         lValorTotal = tempos[i] + lValorTotal;
      return (float)lValorTotal / (fim - inicio + 1);
   }
}

class SecurityManager {
   private StringBuffer sFilteredPolicy;
   private int iApplyCommandsCount;
   private String sApplyCommands[];
   private int iRemoveCommandsCount;
   private String sRemoveCommands[];
   private String ApplyPolicyXSLFileName;
   private String RemovePolicyXSLFileName;
   private String RequirementsFileName;

   public static final int Ok = 0;
   public static final int NotAccepted = -1;
   public static final int PolicyXMLError = -2;
   public static final int TransformXMLError = -3;
   public static final int FileNotFound = -4;
   public static final int URLNotFound = -5;
   public static final int Failure = -6;
   public static final int ExecNotFound = -7;
   public static final int NotAvaliable = -8;
   public static final int OnlyPhase1 = -9;
   public static final int UnknowError = -10;

   public SecurityManager(String ApplyPolicyXSLFileName,
      String RemovePolicyXSLFileName, String RequirementsXSLFileName)
   {
      this.ApplyPolicyXSLFileName = ApplyPolicyXSLFileName;
      this.RemovePolicyXSLFileName = RemovePolicyXSLFileName;
      this.RequirementsFileName = RequirementsXSLFileName;
   }
   public int loadPolicyFile(String PolicyFileName)
   {
      FileInputStream fisPolicy;
      FileInputStream fisPolicy2;
      try {
         fisPolicy = new FileInputStream(PolicyFileName);
         fisPolicy2 = new FileInputStream(PolicyFileName);
      } catch (IOException ioe) {
         // I/O error
         ioe.printStackTrace();
         return SecurityManager.FileNotFound;
      }
      int iErrorCode;
      InputStream isApply;
      InputStream isRemove;
      if(RequirementsFileName!="")
      {
         int res = checkRequirements(fisPolicy, RequirementsFileName);
         if(res != SecurityManager.Ok)
            return res;

         ByteArrayInputStream bais =
            new ByteArrayInputStream(sFilteredPolicy.toString().getBytes());
         ByteArrayInputStream bais2 =
            new ByteArrayInputStream(sFilteredPolicy.toString().getBytes());
         isApply = bais;
         isRemove = bais2;
      }
      else
      {
```

```
      isApply = fisPolicy;
      isRemove = fisPolicy2;
   }
   iErrorCode = Stylizer.XSLTConvert(isApply, ApplyPolicyXSLFileName);
   if(iErrorCode != SecurityManager.Ok)
      return iErrorCode;
   splitCommands(1);

   if(RemovePolicyXSLFileName != "")
   {
      iErrorCode = Stylizer.XSLTConvert(isRemove, RemovePolicyXSLFileName);
      if(iErrorCode != SecurityManager.Ok)
         return iErrorCode;
      splitCommands(2);
   }
   return SecurityManager.Ok;
}

public int loadPolicyFromURL(String sUrl, String RequirementsFileName)
      throws Exception
{
   URL urlWSDL;
   URL urlWSDL2;
   try
   {
      urlWSDL = new URL(sUrl);
      urlWSDL2 = new URL(sUrl);
   }
   catch(Exception e)
   {
      e.printStackTrace();
      return SecurityManager.URLNoteFound;
   }
   int iErrorCode;
   InputStream isApply;
   InputStream isRemove;
   if(RequirementsFileName!="")
   {
      iErrorCode = checkRequirements(urlWSDL.openStream(), RequirementsFileName);
      if(iErrorCode != SecurityManager.Ok)
         return iErrorCode;

      ByteArrayInputStream bais =
         new ByteArrayInputStream(sFilteredPolicy.toString().getBytes());
      ByteArrayInputStream bais2 =
         new ByteArrayInputStream(sFilteredPolicy.toString().getBytes());
      isApply = bais;
      isRemove = bais2;
   }
   else
   {
      isApply = urlWSDL.openStream();
      isRemove = urlWSDL2.openStream();
   }

   iErrorCode = Stylizer.XSLTConvert(isApply, ApplyPolicyXSLFileName);
   if(iErrorCode != SecurityManager.Ok)
      return iErrorCode;
   splitCommands(1);

   if(RemovePolicyXSLFileName != "")
   {
      iErrorCode = Stylizer.XSLTConvert(isRemove, RemovePolicyXSLFileName);
      if(iErrorCode != SecurityManager.Ok)
         return iErrorCode;
```

```
      splitCommands(2);
   }

   return SecurityManager.Ok;
}

private int splitCommands(int Type)
{
   String sAllCommands;
   sAllCommands = Stylizer.sbPoliticas.toString();

   StringTokenizer st = new StringTokenizer(sAllCommands,"\r\n");
   if(Type == 1)
   {
      sApplyCommands = new String[200];
      iApplyCommandsCount = 0;
      while (st.hasMoreTokens())
      {
         sApplyCommands[iApplyCommandsCount] = st.nextToken();
         iApplyCommandsCount++;
      }
   }
   else if(Type == 2)
   {
      sRemoveCommands = new String[200];
      iRemoveCommandsCount = 0;
      while (st.hasMoreTokens())
      {
         sRemoveCommands[iRemoveCommandsCount] = st.nextToken();
         iRemoveCommandsCount++;
      }
   }
   return SecurityManager.Ok;
}

private int checkRequirements(InputStream isIn, String RequirementsFileName)
{
   int iErrorCode = Stylizer.XSLTConvert(isIn, RequirementsFileName);
   if(iErrorCode != SecurityManager.Ok)
      return iErrorCode;
   sFilteredPolicy = Stylizer.sbPoliticas;
   return SecurityManager.Ok;
}

public int applyPolicy()
{
   for(int i = 0; i < iApplyCommandsCount; i++)
   {
      System.out.println("Aplicando política: " + sApplyCommands[i]);
      if(runApp("cmd /D /C " + sApplyCommands[i]) == null)
         return SecurityManager.FileNotFound;

      try {
      Thread.sleep(500);
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
   return SecurityManager.Ok;
}

public int removePolicy()
{
   if(RemovePolicyXSLFileName == "")
   {
```

```
      if(runApp("ipsecpol -u") == null)
         return SecurityManager.FileNotFound;
      return SecurityManager.Ok;
   }
   else
   {
      for(int i = 0; i < iRemoveCommandsCount; i++)
      {
         System.out.println("Removendo política: " + sRemoveCommands[i]);
         if(runApp("cmd /D /C " + sRemoveCommands[i]) == null)
            return SecurityManager.FileNotFound;
      }
      return SecurityManager.Ok;
   }
}

public int verifyIPsecChannel()
{
   String str;
   Vector vNetDiagResult = new Vector();
   BufferedReader br = runApp("netdiag /test:IPSec /v");
   if(br == null)
      return SecurityManager.FileNotFound;
   try
   {
      boolean bAcheiIPsec = false;
      while((str = br.readLine()) != null)
         if((str.length() > 16) &&
            (str.substring(0,16).equalsIgnoreCase("IP Security test")))
         {
            bAcheiIPsec = true;
            break;
         }
      if(!bAcheiIPsec)
      {
         System.out.println("Error while reading information from Netdiag.");
         return SecurityManager.UnknowError;
      }
      vNetDiagResult.add(str);
      while((str = br.readLine()) != null)
         vNetDiagResult.add(str);

      int i = 0;
      while(i < vNetDiagResult.size())
      {
         i++;
      }

   }
   catch(java.io.IOException e)
   {
      System.out.println("Error while reading information about IPsec: " + e);
      return SecurityManager.UnknowError;
   }
   return SecurityManager.Ok;
}

private BufferedReader runApp(String cmdApp)
{
   String str = null;
   Runtime run = Runtime.getRuntime();
   Process process = null;
   try {
      process = run.exec(cmdApp);
      BufferedReader br = new BufferedReader(
```

```
            new InputStreamReader(process.getInputStream()));
         return br;
      }
      catch(java.io.IOException e) {
         System.out.println(
            "Error while executing the process: " + e);
         return null;
      }
   }
}

class Stylizer
{
   static Document document;
   static StringBuffer sbPoliticas;

   public static int XSLTConvert(InputStream Source, String StyleFile)
   {
      DocumentBuilderFactory factory =
         DocumentBuilderFactory.newInstance();

      try {
         File stylesheet = new File(StyleFile);
         DocumentBuilder builder = factory.newDocumentBuilder();
         document = builder.parse(Source);

         TransformerFactory tFactory =
            TransformerFactory.newInstance();
         StreamSource stylesource = new StreamSource(stylesheet);
         Transformer transformer = tFactory.newTransformer(stylesource);

         DOMSource source = new DOMSource(document);

         StringWriter sw = new StringWriter();

         StreamResult result = new StreamResult(sw);//System.out);
         transformer.transform(source, result);

         sbPoliticas = sw.getBuffer();
      } catch (TransformerConfigurationException tce) {
         // Error generated by the parser
         System.out.println ("\n** Transformer Factory error");
         System.out.println("   " + tce.getMessage() );

         // Use the contained exception, if any
         Throwable x = tce;
         if (tce.getException() != null)
            x = tce.getException();
         x.printStackTrace();
         return SecurityManager.TransformXMLError;

      } catch (TransformerException te) {
         // Error generated by the parser
         System.out.println ("\n** Transformation error");
         System.out.println("   " + te.getMessage() );

         // Use the contained exception, if any
         Throwable x = te;
         if (te.getException() != null)
            x = te.getException();
         x.printStackTrace();
         return SecurityManager.TransformXMLError;

      } catch (SAXException sxe) {
         // Error generated by this application
```

```
            // (or a parser-initialization error)
            Exception  x = sxe;
            if (sxe.getException() != null)
               x = sxe.getException();
            x.printStackTrace();
            return SecurityManager.UnknowError;

         } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();
            return SecurityManager.PolicyXMLError;

         } catch (IOException ioe) {
            // I/O error
            ioe.printStackTrace();
            return SecurityManager.FileNotFound;
         }
         return SecurityManager.Ok;
      }
   }
```

ANEXO D – Documento XSL – Transformação de Políticas em Comandos de Aplicação de Políticas

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns="http://www.w3.org/TR/xhtml1/strict">
   <xsl:output method="text" indent="no"/>
   <xsl:template match="definitions">
      <xsl:apply-templates select="SecurityPolicy"/>
   </xsl:template>

   <xsl:template match="SecurityPolicy">
      <xsl:text>echo Aplicando politicas</xsl:text>
      <xsl:apply-templates select="IPsecPolicyGroup"/>
   </xsl:template>

   <xsl:template match="IPsecPolicyGroup">
echo Politica: <xsl:value-of select="@PolicyGroupName"/>
      <xsl:apply-templates select="IKERule"/>
      <xsl:apply-templates select="IPsecRule"/>
      <xsl:text>
ipsecpol</xsl:text>
      <xsl:text> -w REG -p "</xsl:text>
      <xsl:value-of select="@PolicyGroupName"/><xsl:text>" -x </xsl:text>

   </xsl:template>

   <xsl:template match="IKERule">
  echo Regra: <xsl:value-of select="@Name"/>
  <xsl:text>
  ipsecpol</xsl:text>
      <xsl:text> -w REG -p "</xsl:text>
      <xsl:value-of select="../@PolicyGroupName"/><xsl:text>" -r "</xsl:text>
      <xsl:value-of select="@PolicyRuleName"/>
      <xsl:text>" -f</xsl:text>
      <xsl:apply-templates select="SACondition"/>

      <xsl:for-each select="IKEAction">
         <xsl:apply-templates select="."/>
         <xsl:if test="@PolicyActionName">
            <xsl:variable name="ActionName" select="@PolicyActionName"/>
            <xsl:variable name="RuleName" select="../@PolicyRuleName"/>
            <xsl:apply-templates
               select="//IKEAction[attribute::PolicyActionName=$ActionName and
               ../@PolicyRuleName!=$RuleName]"/>
         </xsl:if>
      </xsl:for-each>
   </xsl:template>

   <xsl:template match="SACondition">
      <xsl:apply-templates select="FilterList"/>
      <xsl:if test="@PolicyConditionName">
         <xsl:variable name="ConditionName" select="@PolicyConditionName"/>
         <xsl:variable name="Rule" select=".."/>
         <xsl:apply-templates
```

```
        select="//SACondition[attribute::PolicyConditionName=$ConditionName and
        ..!=$Rule]/FilterList"/>
   </xsl:if>
</xsl:template>

<xsl:template match="FilterList">
   <xsl:for-each select="IPHeadersFilter">
      <xsl:text> </xsl:text>
      <xsl:value-of select="@HdrSrcAddress"/>
      <xsl:if test="@HdrSrcMask!=''">/<xsl:value-of select="@HdrSrcMask"/>
      </xsl:if>
      <xsl:if test="@HdrSrcPortStart!=''">
         :<xsl:value-of select="@HdrSrcPortStart"/>
      </xsl:if>
      <xsl:choose>
         <xsl:when test="../@Direction='4'">+</xsl:when>
         <xsl:otherwise>=</xsl:otherwise>
      </xsl:choose>
      <xsl:value-of select="@HdrDestAddress"/>
      <xsl:if test="@HdrDestMask!=''">/<xsl:value-of select="@HdrDestMask"/>
      </xsl:if>
      <xsl:if test="@HdrDestPortStart!=''">
         :<xsl:value-of select="@HdrDestPortStart"/>
      </xsl:if>
      <xsl:if test="@HdrDestPortStart!='' and HdrProtocolID!=''">
         :<xsl:value-of select="@HdrProtocolID"/>
      </xsl:if>
      <xsl:if test="@HdrDestPortStart!='' and HdrProtocolID=''">
         ::<xsl:value-of select="@HdrProtocolID"/>
      </xsl:if>
   </xsl:for-each>
</xsl:template>

<xsl:template match="IPsecRule">
echo Regra: <xsl:value-of select="@Name"/>
   <xsl:text>
ipsecpol</xsl:text>
   <xsl:text> -w REG -p "</xsl:text>
   <xsl:value-of select="../@PolicyGroupName"/><xsl:text>" -r "</xsl:text>
   <xsl:value-of select="@PolicyRuleName"/><xsl:text>" </xsl:text>
   <xsl:text> -f</xsl:text>
   <xsl:apply-templates select="SACondition"/>
   <xsl:text> -n </xsl:text>
   <xsl:for-each select="IPsecTransportAction">
      <xsl:apply-templates select="."/>
      <xsl:if test="@PolicyActionName">
         <xsl:variable name="ActionName" select="@PolicyActionName"/>
         <xsl:variable name="Rule" select=".."/>
         <xsl:apply-templates
            select="//IPsecTransportAction[attribute::PolicyActionName=$ActionName
            and ..!=$Rule]"/>
      </xsl:if>
   </xsl:for-each>
   <xsl:for-each select="IPsecTunnelAction">
      <xsl:apply-templates select="."/>
      <xsl:if test="@PolicyActionName">
         <xsl:variable name="ActionName" select="@PolicyActionName"/>
         <xsl:variable name="Rule" select=".."/>
         <xsl:apply-templates
            select="//IPsecTunnelAction[attribute::PolicyActionName=$ActionName
            and ..!=$Rule]"/>
      </xsl:if>
   </xsl:for-each>
</xsl:template>
```

```
<xsl:template match="IKEAction">
   <xsl:if test="count(IKEProposal)>0">
      <xsl:text> -a </xsl:text>
      <xsl:for-each select="IKEProposal">
         <xsl:choose>
            <xsl:when test="@AuthenticationMethod='3'">
               <xsl:for-each
select="../../SACondition/AcceptCredentialsFrom/CertificateAuthority/">
                  <xsl:text>CERT:"</xsl:text>
                  <xsl:value-of select="@CADistinguishedName"/>
                  <xsl:text>" </xsl:text>
               </xsl:for-each>
            </xsl:when>
            <xsl:when test="@AuthenticationMethod='6'">KERBEROS</xsl:when>
            <xsl:otherwise>
               <xsl:value-of select="@AuthenticationMethod"/>
            </xsl:otherwise>
         </xsl:choose>
         <xsl:if test="@usePFS='True'">
            <xsl:text> -1p</xsl:text>
         </xsl:if>
         <xsl:if test="@MaxLifetimeSeconds!='' or @MaxLifetimeKilobytes!=''">
            <xsl:text> -1k </xsl:text>
         </xsl:if>
         <xsl:if test="@MaxLifetimeSeconds!=''">
            <xsl:value-of select="@MaxLifetimeSeconds"/>
            <xsl:text>S</xsl:text>
         </xsl:if>
         <xsl:if test="@MaxLifetimeSeconds!='' and @MaxLifetimeKilobytes!=''">
            <xsl:text>/</xsl:text>
         </xsl:if>
         <xsl:if test="@MaxLifetimeKilobytes!=''">
            <xsl:value-of select="@MaxLifetimeKilobytes"/>
            <xsl:text>Q</xsl:text>
         </xsl:if>
      </xsl:for-each>
      <xsl:text> -ls </xsl:text>
      <xsl:for-each select="IKEProposal">
         <xsl:choose>
            <xsl:when test="@CipherAlgorithm='1'">DES</xsl:when>
            <xsl:when test="@CipherAlgorithm='5'">3DES</xsl:when>
         </xsl:choose>
         <xsl:text>-</xsl:text>
         <xsl:choose>
            <xsl:when test="@HashAlgorithm='1'">MD5</xsl:when>
            <xsl:when test="@HashAlgorithm='2'">SHA</xsl:when>
         </xsl:choose>
         <xsl:text>-</xsl:text>
         <xsl:value-of select="@GroupId"/>
      </xsl:for-each>
   </xsl:if>
</xsl:template>

<xsl:template match="IPsecTransportAction">
   <xsl:if test="count(IPsecProposal)>0">
      <xsl:for-each select="IPsecProposal">
         <xsl:sort select="@SequenceNumber"/>
         <xsl:if test="AHTransform[@AHTransformID]">
            <xsl:text>AH[</xsl:text>
               <xsl:choose>
                  <xsl:when test="AHTransform/@AHTransformID='2'">MD5</xsl:when>
                  <xsl:when test="AHTransform/@AHTransformID='3'">SHA</xsl:when>
               </xsl:choose>
            <xsl:text>]</xsl:text>
         </xsl:if>
```

```
        <xsl:if test="AHTransform and ESPTransform">
           <xsl:text>+</xsl:text>
        </xsl:if>
        <xsl:if test="ESPTransform">
           <xsl:text>ESP[</xsl:text>
           <xsl:choose>
              <xsl:when test="ESPTransform/@CipherTransformId='2'">DES</xsl:when>
              <xsl:when test="ESPTransform/@CipherTransformId='3'">3DES</xsl:when>
              <xsl:when test="not(ESPTransform/@CipherTransformId)">
                 NONE</xsl:when>
           </xsl:choose>
           <xsl:text>,</xsl:text>
           <xsl:choose>
              <xsl:when test="ESPTransform/@IntegrityTransformId='1'">
                 MD5</xsl:when>
              <xsl:when test="ESPTransform/@IntegrityTransformId='2'">
                 SHA</xsl:when>
              <xsl:when test="not(ESPTransform/@IntegrityTransformId)">
                 NONE</xsl:when>
           </xsl:choose>
           <xsl:text>]</xsl:text>
        </xsl:if>
     </xsl:for-each>
  </xsl:if>
</xsl:template>

<xsl:template match="IPsecTunnelAction">
     <xsl:for-each select="IPsecProposal">
        <xsl:sort select="@SequenceNumber"/>
        <xsl:if test="AHTransform[@AHTransformID]">
           <xsl:text>AH[</xsl:text>
              <xsl:choose>
                 <xsl:when test="AHTransform/@AHTransformID='2'">MD5</xsl:when>
                 <xsl:when test="AHTransform/@AHTransformID='3'">SHA</xsl:when>
              </xsl:choose>
           <xsl:text>]</xsl:text>
        </xsl:if>
        <xsl:if test="AHTransform and ESPTransform">
           <xsl:text>+</xsl:text>
        </xsl:if>
        <xsl:if test="ESPTransform">
           <xsl:text>ESP[</xsl:text>
           <xsl:choose>
              <xsl:when test="ESPTransform/@CipherTransformId='2'">DES</xsl:when>
              <xsl:when test="ESPTransform/@CipherTransformId='3'">3DES</xsl:when>
              <xsl:when test="not(ESPTransform/@CipherTransformId)">
                 NONE</xsl:when>
           </xsl:choose>
           <xsl:text>,</xsl:text>
           <xsl:choose>
              <xsl:when test="ESPTransform/@IntegrityTransformId='1'">
                 MD5</xsl:when>
              <xsl:when test="ESPTransform/@IntegrityTransformId='2'">
                 SHA</xsl:when>
              <xsl:when test="not(ESPTransform/@IntegrityTransformId)">
                 NONE</xsl:when>
           </xsl:choose>
           <xsl:text>]</xsl:text>
        </xsl:if>
     </xsl:for-each>
  <xsl:apply-templates select="PeerGateway"/>
</xsl:template>

<xsl:template match="PeerGateway">
  <xsl:text> -t </xsl:text>
```

```
      <xsl:value-of select="PeerIdentity"/>
   </xsl:template>
</xsl:stylesheet>
```

ANEXO E – Documento XSL de Requisitos Mínimos

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/xhtml1/strict">
   <xsl:output method="xml" indent="yes"/>

   <xsl:template match="definitions">
      <xsl:copy>
      <xsl:for-each select="@*">
            <xsl:copy/>
      </xsl:for-each>
      <xsl:for-each select="*[not(self::SecurityPolicy)]">
      <xsl:text>
   </xsl:text>
            <xsl:copy-of select="."/>
      </xsl:for-each>
   <xsl:text>
   </xsl:text>
      <xsl:apply-templates select="SecurityPolicy"/>
      <xsl:text>
</xsl:text>
      </xsl:copy>
   </xsl:template>

   <xsl:template match="SecurityPolicy">
      <xsl:copy>
      <xsl:for-each select="@*">
            <xsl:copy/>
      </xsl:for-each>
         <xsl:apply-templates select="IPsecPolicyGroup"/>
      <xsl:text>
   </xsl:text></xsl:copy>
   </xsl:template>

   <xsl:template match="IPsecPolicyGroup">
      <xsl:text>
      </xsl:text>
      <xsl:copy>
      <xsl:for-each select="@*">
            <xsl:copy/>
      </xsl:for-each>
         <xsl:apply-templates select="IKERule"/>
         <xsl:apply-templates select="IPsecRule"/>
      <xsl:text>
      </xsl:text>
      </xsl:copy>
   </xsl:template>

   <xsl:template match="IKERule">
      <xsl:text>
         </xsl:text>
      <xsl:copy>
      <xsl:for-each select="@*">
            <xsl:copy/>
      </xsl:for-each>
```

```
   <xsl:for-each select="*[self::SACondition or
      self::PolicyTimePeriodCondition]">
   <xsl:text>
         </xsl:text>
         <xsl:copy-of select="."/>
   </xsl:for-each>
   <xsl:apply-templates select="IKEAction"/>
   <xsl:apply-templates select="IKERule"/>
   <xsl:text>
      </xsl:text>
   </xsl:copy>
</xsl:template>

<xsl:template match="SACondition">
   <xsl:if test="not(@PolicyConditionName)">
      <xsl:apply-templates select="FilterList"/>
   </xsl:if>
   <xsl:if test="@PolicyConditionName">
      <xsl:variable name="ConditionName" select="@PolicyConditionName"/>
      <xsl:apply-templates select="//SACondition
         [attribute::PolicyConditionName=$ConditionName]/FilterList"/>
   </xsl:if>
</xsl:template>

<xsl:template match="FilterList">
   <xsl:for-each select="IPHeadersFilter">
      <xsl:text> </xsl:text>
      <xsl:value-of select="@HdrSrcAddress"/>
      <xsl:if test="@HdrSrcMask!=''">/<xsl:value-of select="@HdrSrcMask"/>
      </xsl:if>
      <xsl:if test="@HdrSrcPortStart!=''">:<xsl:value-of
         select="@HdrSrcPortStart"/>
      </xsl:if>
      <xsl:choose>
         <xsl:when test="../@Direction='4'">+</xsl:when>
         <xsl:otherwise>=</xsl:otherwise>
      </xsl:choose>
      <xsl:value-of select="@HdrDestAddress"/>
      <xsl:if test="@HdrDestMask!=''">
         /<xsl:value-of select="@HdrDestMask"/>
      </xsl:if>
      <xsl:if test="@HdrDestPortStart!=''">:<xsl:value-of
         select="@HdrDestPortStart"/>
      </xsl:if>
      <xsl:if test="@HdrDestPortStart!='' and
         HdrProtocolID!=''">:<xsl:value-of select="@HdrProtocolID"/>
      </xsl:if>
      <xsl:if test="@HdrDestPortStart!='' and
         HdrProtocolID=''">::<xsl:value-
         of select="@HdrProtocolID"/>
      </xsl:if>
   </xsl:for-each>
</xsl:template>

<xsl:template match="IPsecRule">
   <xsl:text>
      </xsl:text>
   <xsl:copy>
   <xsl:for-each select="@*">
         <xsl:copy/>
   </xsl:for-each>
   <xsl:for-each select="*[self::SACondition or
      self::PolicyTimePeriodCondition]">
   <xsl:text>
         </xsl:text>
```

```
        <xsl:copy-of select="."/>
    </xsl:for-each>
    <xsl:apply-templates select="IPsecTransportAction"/>
    <xsl:apply-templates select="IPsecTunnelAction"/>
    <xsl:apply-templates select="IPsecRule"/>
    <xsl:text>
       </xsl:text>
    </xsl:copy>
</xsl:template>

<xsl:template match="IKEAction">
    <xsl:text>
        </xsl:text>
    <xsl:copy>
    <xsl:for-each select="@*">
        <xsl:copy/>
    </xsl:for-each>
    <xsl:for-each select="*[self::IKEProposal]">
       <xsl:if test="@CipherAlgorithm='5'">
          <xsl:text>
             </xsl:text>
             <xsl:copy-of select="." >
          </xsl:if>
    </xsl:for-each>
    <xsl:text>
        </xsl:text>
    </xsl:copy>
</xsl:template>

<xsl:template match="IPsecTransportAction">
    <xsl:text>
        </xsl:text>
    <xsl:copy>
    <xsl:for-each select="@*">
        <xsl:copy/>
    </xsl:for-each>
    <xsl:for-each select="*">
       <xsl:if test="AHTransform/@AHTransformID='3' and
          ESPTransform/@CipherTransformId='3'">
          <xsl:text>
             </xsl:text>
             <xsl:copy-of select="."/>
          </xsl:if>
    </xsl:for-each>
    <xsl:text>
        </xsl:text>
    </xsl:copy>
</xsl:template>

<xsl:template match="IPsecTunnelAction">
      <xsl:text>
         </xsl:text>
    <xsl:copy>
    <xsl:for-each select="@*">
        <xsl:copy/>
    </xsl:for-each>
    <xsl:for-each select="*">
       <xsl:if test="AHTransform/@AHTransformID='SHA'  and
          ESPTransform/@CipherTransformId='3DES'">
          <xsl:text>
             </xsl:text>
             <xsl:copy-of select="."/>
          </xsl:if>
    </xsl:for-each>
    <xsl:text>
```

```
            </xsl:text>
        <xsl:copy-of select="PeerGateway"/>
        <xsl:text>
              </xsl:text>
        </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```