MARCELLO HENRIQUE TOZONI MILANEZ

USO DE CADEIAS DE AUTORIZAÇÃO SPKI NA SEGURANÇA DE REDES SEM FIO IEEE 802.11

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. Carlos Alberto Maziero

CURITIBA

2003

Milanez, Marcello Henrique Tozoni
Uso de Cadeias de Autorização SPKI na Segurança de Redes sem Fio IEEE 802.11. Curitiba, 2003. 125p.

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

1. Segurança 2. Autorização 3. SPKI 4. IEEE 802.11. I.Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática Aplicada II-t

Agradecimentos

Aos meus pais Wilson e Maria Isabel e ao meu irmão Gabriel por todo apoio e incentivo.

Ao meu orientador prof. Carlos Maziero por sua dedicação e colaboração.

Sumário

Lista de Figuras

Resumo

As redes sem fio (*wireless*) apesar de muito recentes, têm conquistado rapidamente o mercado por proporcionarem algumas vantagens em relação às redes tradicionais (com cabos), como a mobilidade e a facilidade de criação de redes temporárias.

Entretanto, este crescimento propiciou um aumento no interesse de invasores em obter acesso a estas redes para capturar informações sigilosas, podendo conseguir até mesmo com certa facilidade, dependendo do tipo de segurança utilizada. A invasão de uma rede sem fio pode ser muito prejudicial, pois muitas empresas as utilizam como extensões de suas redes fixas e conseqüentemente um invasor pode acabar acessando toda a rede corporativa.

Por isso, a segurança tornou-se um aspecto vital em ambientes de redes sem fio. No caso das redes padrão IEEE 802.11, mesmo a utilização da autenticação *Shared Key* juntamente com a privacidade fornecida pelo WEP não é suficiente e, por isso, outras formas de segurança devem ser desenvolvidas e pesquisadas.

Tentando melhorar a segurança oferecida pelo padrão IEEE 802.11 foi elaborada esta dissertação, propondo para isso utilizar os protocolos EAP e TLS e a infra-estrutura de chave pública SPKI. Para proporcionar uma melhor flexibilidade e agilidade no processo de autorização neste tipo de rede, foi utilizada SPKI que através de certificados delegáveis permite que as estações móveis (*peers*) sejam autorizadas com maior rapidez.

Palavras-Chave: autorização, SPKI, IEEE 802.11, certificados, segurança.

Abstract

Wireless networks offer several advantages over classic wired networks, like mobility and the ability to easily establish temporary networks. Consequently, the market is well accepting them.

In the meantime, this rapid growing also got the attention from network intruders. Depending on the technology being used, wireless networks security can be easy to break-in. An intrusion in a wireless network is potentially very dangerous, as companies generally use them are extensions of the corporate networks. In this case, an intruder could have access to the entire network.

Thus, security has become a vital concern in wireless networks. When considering IEEE 802.11 –based networks, the generally used shared key authentication and WEP privacy are not sufficient, and other security improvements should be provided.

This dissertation presents a proposal to improve security on IEEE 802.11 networks. It proposes to combine the Extensible Authentication Protocol (EAP) to the Simple Public Key Infrastructure (SPKI). The joint usage of both technologies allows to achieve a better flexibility and speed in the authorization procedure.

Keywords: security, authorization, SPKI, IEEE 802.11.

Capítulo 1

Introdução

Durante os últimos anos, a crescente necessidade de comunicação do ser humano tem impulsionado o desenvolvimento de novas tecnologias que possibilitem o acesso a informações, independentemente do local e da situação em que a pessoa esteja, como por exemplo, em locais isolados ou em deslocamento.

Dentre estas novas tecnologias podem ser citados os satélites, e aparelhos pessoais portáteis como telefones celulares, PDAs e laptops. Para integrá-los, surgiu a necessidade da criação de redes capazes de possibilitar a comunicação entre estes aparelhos.

Foi neste contexto que surgiram as redes *wireless*, visando possibilitar que aparelhos portáteis possam comunicar-se entre si ou com um elemento intermediador que permita a sua interação com uma rede fixa. Como exemplo de padrão para redes *wireless* pode-se citar o IEEE 802.11.

Com a sua disseminação, as redes *wireless* têm possibilitado uma grande melhoria no acesso a informações, principalmente em relação à rapidez e à mobilidade. Entretanto, juntamente com os benefícios vieram as desvantagens, sendo uma das principais delas a falta de segurança, devido principalmente às suas novas características como a utilização de ondas de rádio ao invés de cabos, o que acaba facilitando a escuta e dificultando ainda mais a proteção das informações quando comparadas com as redes fixas.

Visando melhorar a segurança das redes *wireless* começaram a ser pesquisadas e desenvolvidas possíveis soluções para minimizar o problema. No caso específico das redes IEEE 802.11 iniciou-se pelo WEP (*Wired Equivalent Privacy*) que se baseia no compartilhamento de chave simétrica entre os dispositivos comunicantes e funciona nas camadas de baixo nível. Entretanto, como esta solução acabou proporcionando uma proteção

fraca, optou-se por uma abordagem das camadas superiores (a partir da camada de rede) a fim de complementar a segurança proporcionada pelo WEP. Foi a partir de então que passaram a ser utilizados protocolos como o EAP (*Extensible Authentication Protocol*) e a ele associado o TLS (*Transport Level Security*), que possibilitaram a autenticação dos elementos através de certificados X.509.

Baseado nesta infra-estrutura de segurança existente para redes IEEE 802.11 foi desenvolvida esta dissertação, visando apresentar uma alternativa segura e que proporcione uma maior flexibilidade no processo de autorização através do uso de certificados SPKI (*Simple Public Key Infrastructure*), além de fortalecer o aspecto de segurança pelo acréscimo de autorização ao processo de autenticação do EAP.

Esta dissertação está estruturada da seguinte forma: o capítulo 2 apresenta dois padrões para redes *wireless* Bluetooth e IEEE 802.11, descrevendo as suas principais características como funcionamento e associação entre seus elementos. No capítulo 3 são discutidos os aspectos de segurança, descrevendo protocolos de segurança como EAP e TLS, e a segurança proporcionada pelos padrões IEEE 802.11 e IEEE 802.1X. No capítulo 4 é apresentada a infra-estrutura SPKI, descrevendo as suas principais características como os certificados SPKI e a elaboração das cadeias de certificados utilizadas para comprovar autorização. No capítulo 5 é discutida a proposta desenvolvida nesta dissertação, descrevendo o escopo e os objetivos e explicando os seus três roteiros de funcionamento. No capítulo 6 é apresentada a descrição detalhada de como foi realizada a validação informal da proposta utilizando uma simulação em Java. No capítulo 7 são apresentadas as conclusões finais, incluindo as contribuições e limitações da proposta abordada na dissertação e trabalhos futuros.

Capítulo 2

Padrões para Redes Sem Fio

Para suprir algumas das necessidades atuais do ser humano, como ter acesso a informações mesmo quando estiver em deslocamento e a possibilidade de criar redes temporárias (*ad hoc*) de maneira simplificada, surgiram as redes sem fio (*wireless).*

Estas redes nos primórdios enfrentaram problemas para sua utilização, como o alto custo dos equipamentos e falta de uma padronização. Com o aperfeiçoamento das tecnologias envolvidas, o conseqüente barateamento dos equipamentos e a criação de padrões para o seu funcionamento, as redes *wireless* conseguiram popularizar-se nos últimos anos, se tornando presentes tanto em ambientes corporativos quanto domésticos.

Neste capítulo, serão apresentados dois padrões para redes *wireless*: *Bluetooth* e IEEE 802.11. Estes foram escolhidos por serem padrões que possuem especificações bem definidas e estarem em fase de expansão, tornando-se disponíveis em um número crescente de produtos no mercado.

Vale ressaltar que apesar de os dois padrões utilizarem a mesma faixa de freqüência (2,4 GHz) eles possuem uma diferença importante: o *Bluetooth* está direcionado para redes pequenas e de pouca abrangência de sinal (aproximadamente 10 metros) denominadas PANs (*Personal Area Networks*) enquanto o IEEE 802.11 está focado em redes de maior tamanho e abrangência denominadas LANs (*Local Area Networks*).

2.1. Bluetooth

O nome *Bluetooth* foi inspirado de um rei da Dinamarca no século X, chamado *Harald Bluetooth,* responsável pela unificação da Dinamarca com a Noruega. Este nome foi então

escolhido porque achou-se que seria apropriado para simbolizar o princípio básico da tecnologia: integração de aparelhos diferentes (assim como o rei *viking* integrou Dinamarca e Noruega).

O *Bluetooth* surgiu com o intuito de definir um padrão global que possibilitasse a comunicação sem cabos entre aparelhos *wireless*, independente de fabricante e do tipo do aparelho (PDA, celular, *laptop*) e deles com aparelhos fixos (*desktops*, impressoras, *scanners*).

O desenvolvimento da tecnologia se iniciou a partir de um estudo feito pela *Ericsson* em 1994 para identificar a viabilidade de criar uma interface de rádio para a comunicação entre celulares e outros aparelhos como *laptops*, dispensando a utilização de cabos.

Anos mais tarde, em 1998 foi formado o *Bluetooth* SIG (*Special Interest Group*) tornando-se o órgão responsável por monitorar o desenvolvimento técnico, definir um padrão global aberto para a tecnologia, bem como garantir total interoperabilidade entre os aparelhos de diversos fabricantes que utilizem o mesmo *Profile* (especificação de como utilizar a pilha de protocolos *Bluetooth* a fim de garantir interoperabilidade). Atualmente, o SIG é formado por grandes empresas da área de informática e telecomunicações como: 3Com, Ericsson, IBM, Intel, Lucent, Microsoft, Motorola, Nokia, Toshiba, além de outras companhias associadas.

De um modo geral, *Bluetooth* trata-se de uma tecnologia que está se transformando em um padrão aberto global, criada com o objetivo de substituir cabos na interconexão de aparelhos pessoais a curtas distâncias (geralmente de 10 a 100 metros), possibilitando a interoperabilidade entre eles e o estabelecimento de redes *ad hoc* (temporárias).

2.1.1. Objetivos

Os principais objetivos a serem atingidos pela tecnologia *Bluetooth* são os seguintes:

- Eliminar cabos e fios nas comunicações entre diferentes tipos de aparelhos tanto fixos quanto móveis, independentemente de fabricantes;
- Possibilitar a comunicação tanto de dados quanto de voz;
- Oferecer a possibilidade de formação de redes *ad hoc*, bem como fornecer sincronização entre os aparelhos pessoais.

2.1.2. Características Técnicas

A transmissão entre aparelhos é feita via sinais de rádio, através de um pequeno chip *Bluetooth*. A especificação do *Bluetooth* define uma abrangência de 10 a 100 metros para o *link* de rádio nas transmissões de voz ou dados.

O sinal de rádio opera em uma faixa não licenciada chamada ISM (*Industrial, Scientific and Medical*) que varia de 2,4 a 2,48 GHz, utilizando sinal *full-duplex* e *frequency hopping* (saltos de freqüência) de até 1600 *hops*/segundo. O sinal varia entre 79 freqüências em um intervalo de 1 MHz, fornecendo uma certa imunidade a interferências.

2.1.3. Formação de Redes

Quando unidades *Bluetooth* estão na mesma área de abrangência de sinal, elas podem iniciar conexões *ad hoc* ponto-a-ponto e/ou ponto-a-multiponto, onde unidades podem ser adicionadas ou desconectadas dinamicamente. Duas ou mais unidades que compartilhem o mesmo canal formam uma *piconet*.

Quando *piconets* são conectadas formam uma *scatternet*, com o propósito de permitir a comunicação e facilitar a troca de dados entre elas. Se várias *piconets* estiverem dentro da mesma área de abrangência, elas podem funcionar sem grandes problemas com interferência, pois cada uma delas é estabelecida com um canal de *frequency hopping* diferente. Dentro de uma *piconet*, todos os participantes são sincronizados para o mesmo canal.

Com o objetivo de regular o tráfego no canal, uma das unidades participantes torna-se o mestre da *piconet*, enquanto as outras unidades tornam-se escravos. O mestre é o responsável por estabelecer e controlar a comunicação, selecionando a freqüência de *hops* que os dispositivos utilizarão. Esta freqüência de *hops* é determinada por um algoritmo presente no mestre e posteriormente informada aos escravos para que possam receber os dados. Por utilizar *frequency hopping* para transmissão, somente os dispositivos sincronizados serão capazes de participar da comunicação, pois os dados são quebrados em pequenos pacotes e enviados aos dispositivos receptores utilizando-se até 79 freqüências diferentes, podendo variar até 1600 vezes por segundo, o que dificulta “escutas” na comunicação e constitui-se também numa forma de segurança.

De acordo com [20], até sete escravos podem comunicar-se ativamente com um mestre.

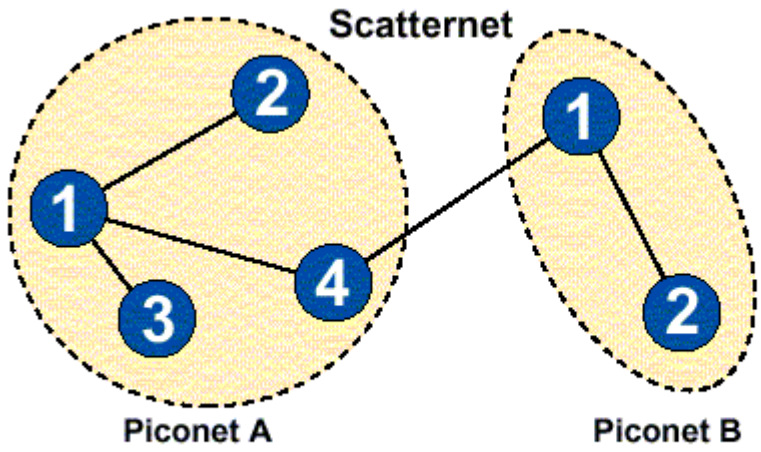

Figura 2.1 - Piconets e Scatternet (obtido de [20])

2.1.4. Funcionamento básico

Os módulos *Bluetooth* possuem *transceivers* responsáveis pela transmissão e recepção de dados utilizando ondas de rádio. Estes módulos são capazes de detectar outros dispositivos *Bluetooth* quando estão em sua área de abrangência e então estabelecer contato e formar uma rede temporária (*piconet*). Antes da transmissão de dados entre os dispositivos, deve ser estabelecida uma sessão de rede entre eles. Quando é criada uma *piconet* ou quando um novo elemento for inserido a ela, é necessário que seja estabelecida esta sessão baseada em dois parâmetros: endereço MAC (*Media Access Control*), pré-programado pelo fabricante e um número PIN (*Personal Identification Number)* atribuído pelo usuário.

Para que dispositivos *Bluetooth* possam se comunicar, eles precisam se conhecer e se identificar. Se ambos dispositivos pertencem à mesma *piconet*, a comunicação é instantânea. Caso contrário, o usuário precisa aprovar manualmente para que se inicie a comunicação. Isto acontece da seguinte forma: quando um dispositivo *Bluetooth* desconhecido entra na área de abrangência de outro, o segundo perguntará ao usuário se ele permite o estabelecimento de uma sessão de rede com primeiro. Se a comunicação entre os dispositivos acontece pela primeira vez, o usuário insere o mesmo PIN (geralmente um número de 4 dígitos) em ambos e a partir deste, dos MACs e de algoritmos existentes nos aparelhos serão definidas as chaves utilizadas na comunicação. A partir de então, o usuário pode definir que o mesmo PIN seja

utilizado em comunicações futuras entre os mesmos dispositivos ou que um novo PIN seja definido para cada nova comunicação.

Após o estabelecimento (criação) de uma *piconet*, o usuário pode configurar os dispositivos para que quando estiverem novamente na mesma área de abrangência, criem uma nova *piconet* sem a intervenção do usuário.

Devido ao seu aspecto dinâmico, um elemento pode entrar e sair de uma *piconet* (após conhecido e identificado) com facilidade, sem que sejam necessárias mudanças de configuração e interferência do usuário.

2.1.5. Segurança em Bluetooth

A segurança proporcionada pela tecnologia *Bluetooth* se baseia em três aspectos principais: autenticação, autorização e privacidade.

Complementarmente, segundo [20], pode-se considerar que *frequency hopping* (alterações da freqüência de transmissão de dados) e a limitada abrangência das transmissões dos dispositivos *Bluetooth* (aproximadamente 10 metros) também auxiliam na segurança das comunicações, principalmente para evitar *eavesdropping* (escuta dos dados em transmissão por um indivíduo externo à comunicação).

A **autenticação** baseia-se em chaves (*link keys*) e um mecanismo de *challenge-response,* servindo principalmente para evitar *spoofing* (falsificação de endereço para o recebimento de dados em uma comunicação por um indivíduo externo a ela). Após o processo de autenticação, pode existir um processo de **autorização**. Um caso prático que envolve a autorização em *Bluetooth* é o descrito em [32], onde se sugere a utilização de um *Security Manager* que acessa bases de dados para verificar se o dispositivo anteriormente autenticado pode ter acesso ao serviço solicitado.

A **privacidade** é garantida através de criptografia, especialmente na troca de informações sigilosas, evitando principalmente *eavesdropping*. A criptografia baseia-se em chaves criptográficas simétricas geradas a partir das *link keys,* conforme descrito em [5].

2.1.6. Elementos de Segurança

Para garantir a segurança na camada de link, o *Bluetooth* utiliza-se de quatro elementos [5]:

- BD_ADDR (*Bluetooth Device Address*): endereço públicos de 48 bits, definido pela IEEE, e único para cada unidade *Bluetooth*.
- *Private Authentication Key*: número secreto de 128 bits utilizado para a autenticação do dispositivo.
- *Private Encryption Key*: número secreto que pode variar de 8 a 128 bits, geralmente derivado da *authentication key*.
- RAND: número de 128 bits gerado a partir de um processo aleatório ou pseudo-aleatório na unidade *Bluetooth*.

Chaves

A *link key* é uma chave formada por um número aleatório de 128 bits, compartilhado por dois ou mais dispositivos, utilizada como base para todas as transações seguras entre eles. Ela é utilizada no processo de autenticação e serve como parâmetro para a geração da *encryption key*.

Quanto à sua duração, as *link keys* podem ser:

- Temporárias: limitadas ao tempo de vida da sessão. Não podem ser reutilizadas posteriormente.
- Semi-permanente: é armazenada em memória não volátil e poderá ser usada em sessões posteriores.

Quanto aos tipos, as *link keys* podem ser:

- *Encryption key* (*KC)*: é a chave de criptografia derivada da *link key* atual. A *encription key* foi separada da *authentication key* com o objetivo de permitir o uso

de pequenas chaves de criptografia sem enfraquecer os procedimentos de autenticação.

- *Unit key (KA)*: é a chave gerada na unidade *Bluetooth* durante a sua instalação. É raramente alterada e não depende de parâmetros externos a unidade.

- *Combination key (KAB)*: é uma chave derivada de informações de duas unidades quaisquer, genericamente chamadas *A* e *B*. A *combination key* é derivada para cada nova combinação de duas unidades *Bluetooth*.

- *Temporary master key (Kmaster)*: é uma *link key* temporária utilizada somente para uma sessão. Pode ser utilizada quando um mestre quer enviar dados criptografados para mais de dois escravos usando a mesma *encryption key*.

- *Initialization key (Kinit)*: é a *link key* utilizada durante o processo de inicialização quando as *combination* ou *unit keys* ainda não foram geradas ou quando a *link key* foi perdida. Ela é gerada a partir de um número aleatório, código PIN e BD_ADDR. Esta chave é utilizada somente durante a inicialização.

O PIN pode ser um número fixo fornecido pela unidade *Bluetooth* ou um número escolhido pelo usuário e inserido nas unidades que deverão se comunicar. O tamanho do número PIN pode variar de 1 a 16 bytes.

2.2. IEEE 802.11

Uma *wireless* LAN (WLAN) é um sistema de transmissão desenvolvido para fornecer acesso à rede (independentemente de localização) entre dispositivos de computação, utilizando sinais de rádio ao invés de utilizar cabos. No ambiente corporativo, as WLANs são geralmente utilizadas como uma “extensão” das redes convencionais, fornecendo o link final entre elas e um grupo de computadores clientes, permitindo aos usuários acesso *wireless* aos recursos e serviços da rede corporativa.

Com o passar do tempo, a utilização de *wireless* LANs foi crescendo, o que gerou a necessidade de criar de uma padronização para elas. Então, com o objetivo de permitir a compatibilidade de produtos para redes sem fio de diferentes fabricantes, o IEEE (*Institute of Electrical and Electronics Engineers*) definiu em 1997 um padrão para WLANs chamado 802.11, abrangendo inicialmente transmissões de dados entre 1 Mbps e 2 Mbps.

Entretanto, após a padronização um outro fator crítico surgiu para limitar o crescimento da utilização de WLANs: o baixo *throughput*, pois as taxas de transmissão de dados suportadas pelo padrão 802.11 eram muito baixas para as exigências do ambiente corporativo. Identificada esta necessidade, o IEEE definiu em setembro de 1999 o padrão 802.11b (conhecido como 802.11 *High Rate*) que suporta a transmissão de até 11 Mbps. A partir de então, as WLANs continuaram evoluindo e atualmente podem transmitir a até 54 Mbps, atingindo performances e *throughputs* comparáveis as *wired Ethernets*. A transmissão a 54 Mbps foi recentemente disponibilizada na especificação 802.11a (operando na freqüência de 5 GHz) e está sendo proposta na 802.11g (operando em 2,4 GHz) que deve ser concluída no segundo semestre de 2003. Arquitetura, características e serviços do 802.11b são definidos pelo padrão original 802.11. A especificação do 802.11b afeta somente a camada física, adicionando taxas de dados mais altas e conectividade mais robusta.

Independentemente da estrutura de padronizações da IEEE, os principais fabricantes de equipamentos para WLANs uniram-se para formar a WECA (*Wireless Ethernet Compatibility Alliance*), com o objetivo de certificar a interoperabilidade entre produtos de fabricantes diferentes, bem como garantir a compatibilidade dos produtos com o padrão 802.11b. Entre os membros da WECA, estão presentes fabricantes de semicondutores para WLANs, provedores de WLANs, fornecedores de sistemas de computador e desenvolvedores de software, como: 3Com, Aironet, Apple, Breezecom, Cabletron, Compaq, Dell, Fujitsu, IBM, Interasil, Lucent Technologies, No Wires Needed, Nokia, Samsung, Symbol Technologies, Wayport, Zoom.

Além da padronização elaborada pela IEEE e da garantia de interoperabilidade fornecida pela WECA, outro ponto impulsionador da utilização das WLANs tem sido as vantagens proporcionadas, incluindo:

- Mobilidade, com o conseqüente aumento de produtividade devido ao acesso a informações em tempo real, independentemente de localização e das restrições

quanto à movimentação imposta pelos cabos, tornando mais rápida e eficiente a tomada de decisão.

- Facilidade para instalação, principalmente em locais de difícil acesso para a passagem de cabos, como construções antigas.

- Reduzido custo de propriedade, principalmente em ambientes dinâmicos que necessitem de modificações freqüentes, devido aos custos relativamente baixos de instalação por dispositivo e usuário.

Em relação à pilha de protocolos, o IEEE 802.11 está focado nas duas primeiras camadas do modelo OSI (camada Física e camada de Enlace).

Quanto aos equipamentos utilizados em uma WLAN, o padrão 802.11 define dois tipos: *station* (estação *wireless*), geralmente um PC ou *laptop* equipado com um NIC (*Network Interface Card*) *wireless,* e um *access point* (AP), que atua como uma ponte entre as redes *wireless* e *wired*. Um *access point* geralmente é composto por um *transceiver* de rádio, uma interface de rede *wired* e um *bridging* software (software para realizar a ponte entre as redes *wireless* e *wired*) que deve estar de acordo com o *bridging standard* 802.1d.

Para a comunicação entre dispositivos, podem ser formadas duas topologias básicas: *ad hoc mode*, onde as estações comunicam-se diretamente entre si; *infraestructure mode*, onde existe a presença de um AP intermediando as comunicações.

Para a transmissão do meio físico, o padrão IEEE 802.11 especifica três tecnologias diferentes: infravermelho ou rádio utilizando FHSS (*Frequency-Hopping Spread Spectrum*) ou DSSS (*Direct Sequence Spread Spectrum*). As duas tecnologias de rádio operam na banda não licenciada de 2.4 GHz chamada ISM (*Industrial, Scientific and Medical*).

Para a utilização de acesso múltiplo ao meio, o padrão IEEE 802.11 define a utilização do protocolo CSMA/CA (*Carrier Sense Multiple Access / Collision Avoidance*), onde cada estação deve “ouvir” o meio antes de transmitir a fim de evitar colisões.

O padrão de endereçamento utilizado é o de 48 bits da IEEE 802 a fim de manter a compatibilidade com os demais padrões da família IEEE 802.

A segurança no IEEE 802.11 está baseada em autenticação e privacidade, podendo operar em dois modos distintos: *Open System* (onde somente autenticação é possível) e

Shared Key (onde se pode utilizar autenticação e privacidade), sendo que a última utiliza WEP (*Wired Equivalent Privacy*) como mecanismo de privacidade. A descrição do funcionamento da segurança no ambiente IEEE 802.11 será abordada posteriormente no capítulo 3 (segurança).

2.2.1. Modos de operação

O padrão 802.11 define dois modos de funcionamento: *infrastructure mode* e *ad hoc mode*.

No *infrastructure mode*, a rede *wireless* é composta por pelo menos um *access point* conectado a rede *wired* e um conjunto de estações *wireless* comunicando-se com ele. Esta configuração é chamada BSS (*Basic Service Set*). Um conjunto de duas ou mais BSSs é chamado de ESS (*Extended Service Set*). As WLANs corporativas que necessitem acessar serviços da LAN *wired* (como servidores de arquivos, impressoras ou links de Internet) precisam operar neste modo.

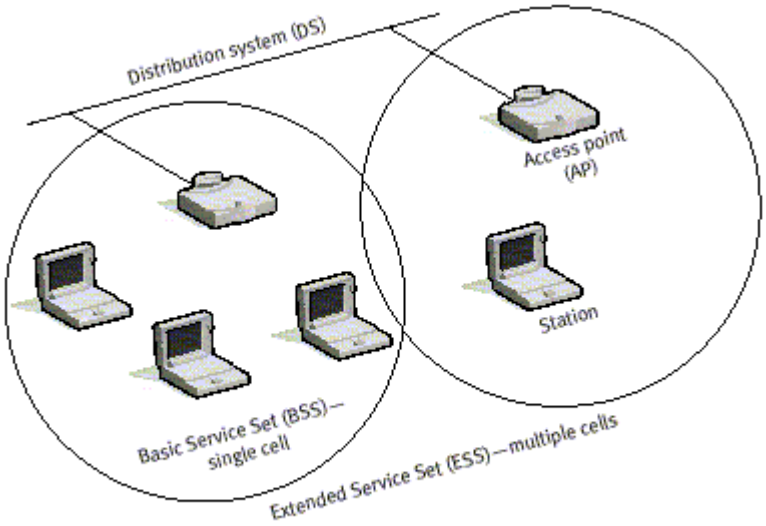

Figura 2.2 - Infrastructure Mode (obtido de [1])

No *ad hoc mode* (também chamado *peer-to-peer mode* ou IBSS – *Independent Basic Service Set*) a rede é composta por um conjunto de estações *wireless* padrão IEEE 802.11 que se comunicam diretamente entre si sem utilizar um *access point* ou conexões à rede *wired*.

Este modo é utilizado quando se deseja estabelecer uma rede *wireless* rapidamente em locais onde não exista uma infra-estrutura para ela, como em reuniões e aeroportos.

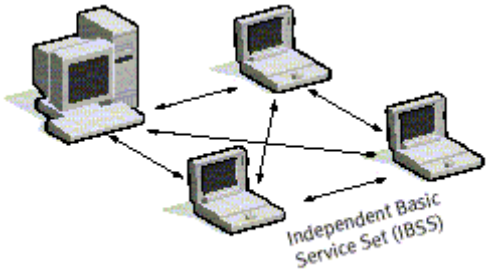

Figura 2.3 - Ad Hoc Mode (obtido de [1])

2.2.2. Camadas

A arquitetura do padrão 802.11 baseia-se nas duas primeiras camadas da pilha de protocolos OSI: camada Física e camada de Enlace, conforme apresentado na figura a seguir.

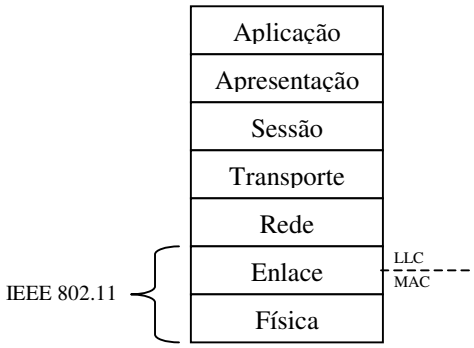

Figura 2.4 - IEEE 802.11 e os protocolos OSI

Camada Física do IEEE 802.11

Os três tipos diferentes de camadas físicas definidas no padrão IEEE *802.11* incluem as técnicas de *spread-spectrum radio* FHSS *(Frequency-Hopping Spread Spectrum*) e DSSS (*Direct Sequence Spread Spectrum)* e uma especificação de infravermelho, conforme definido em [28]. A freqüência de 2.4 GHz, utilizada pela IEEE 802.11 chamada de ISM (Industrial, Scientific and Medical) é reconhecida por agências internacionais (FCC nos EUA, ETSI na Europa e MKK no Japão) como livre de licença para operações de rádio.

Camada de Enlace do IEEE 802.11

A sua camada de Enlace é composta por duas subcamadas: LLC (*Logical Link Control*) e MAC (*Medium Access Control*). Utiliza-se do mesmo 802.2 LCC e endereçamento de 48 bits das outras LANs no padrão 802.

O 802.11 MAC tem a concepção muito parecida ao 802.3 (padrão *wired Ethernet*). Como diferencial, o IEEE 802.11 utiliza um protocolo chamado CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*) ou DCF (*Distributed Coordination Function*). Este protocolo tenta evitar colisões usando pacotes de confirmação (*ACKs*) que são enviados pela estação receptora a fim de confirmar que o pacote de dados chegou intacto.

O CSMA/CA funciona da seguinte forma: quando uma estação deseja transmitir, ela tenta detectar se existe alguma transmissão sendo realizada, e caso não exista, ela espera um período de tempo aleatório e se o meio ainda estiver livre, ela transmite. Se o pacote for recebido intacto, a estação receptora gera um ACK *frame* que quando recebido pelo emissor garante que a transmissão foi efetuada com sucesso. Se o ACK *frame* não for recebido pelo emissor, devido ao pacote de dados original ou o ACK não ter sido recebido intacto, assume-se que ocorreu uma colisão e o pacote de dados é retransmitido após uma espera aleatória.

Outro diferencial da camada MAC em ambiente *wireless* é existência do chamado "*hidden node*" (nó oculto), onde duas estações situadas em lados opostos do *access point* podem ambas "ouvir" a atividade do *access point*, mas não uma da outra, devido principalmente à distância entre elas ou a alguma obstrução. Para resolver este problema, o padrão IEEE 802.11 especificou um protocolo opcional para a camada MAC chamado RTS/CTS (*Request to Send/Clear to Send*). Quando este protocolo é utilizado, a estação emissora transmite um RTS ao *access point* e aguarda um CTS como resposta. As outras estações da rede que puderem ouvir o *access point* também receberão o CTS, a fim de serem informadas que o meio estará sendo utilizado por outra estação e terão que aguardar para poder transmitir. Isto permite que a estação emissora possa transmitir e receber os *ACKs* sem que ocorram colisões. Por adicionar um custo a rede pela reserva temporária do meio, o RTS/CTS é utilizado somente para grandes pacotes, para os quais a retransmissão pode ser cara sob o ponto de vista de utilização da banda disponível.

2.2.3. Associações

A camada MAC do padrão 802.11 também é responsável pela forma como uma estação associa-se a um *access point*. A associação por parte do cliente ocorre resumidamente da seguinte forma: a estação ao entrar na área de abrangência de sinal de um ou mais *access points* pode escolher um deles para associar-se (passando a fazer parte de um *Basic Service Set* – BSS), levando em consideração principalmente à força do sinal e a taxa de erros de pacotes. Após escolher o AP e ser aceito por ele, o cliente ajusta o canal de rádio para o qual o *access point* está configurado. Periodicamente, a estação avalia todos os canais 802.11 em sua área de abrangência para identificar se algum outro AP pode fornecer serviços com um melhor desempenho. Caso esta identificação aconteça, ele pode associar-se a este outro *access point*, ajustando-se ao canal de rádio para o qual o AP está configurado.

As mudanças de associações ocorrem principalmente devido a: movimentação da estação *wireless* para uma posição mais distante do *access point*, com um conseqüente enfraquecimento do sinal; grande aumento do tráfego de rede no *access point*, sendo esta associação chamada de "*load balancing*", objetivando uma distribuição de carga mais eficiente na infra-estrutura *wireless* disponível da WLAN.

Vale salientar que para uma estação associar-se a um AP é necessário que esteja autenticada a ele, utilizando um dos tipos de autenticação da camada de Enlace disponíveis para redes IEEE 802.11 (*Open System* ou *Shared Key*), conforme descrito posteriormente no capítulo 3 (segurança). Caso a estação não tenha sido autenticada pelo AP, ela deve realizar o processo de autenticação antes de tentar se associar a ele.

2.3. Conclusão

Este capítulo apresentou dois dos principais padrões disponíveis no mercado para redes *wireless*: Bluetooth para PANs e IEEE 802.11 para LANs, descrevendo suas características e formas de associações entre os seus elementos. Foram também descritos os componentes básicos de segurança em uma rede Bluetooth.

A escolha do padrão IEEE 802.11 para o desenvolvimento do trabalho foi feita em função de algumas de suas características como existência de uma padronização definida e amplamente utilizada, crescente utilização em WLANs e a fraca segurança proporcionada

pelas camadas de baixo nível (Física e Enlace) definidas no padrão, o que possibilita um vasto campo para o desenvolvimento de pesquisas e a proposição de melhorias.

Capítulo 3

Segurança em Redes IEEE 802.11

Os aspectos de segurança têm despertado uma atenção especial, principalmente nos últimos anos, em função da disseminação de novas tecnologias de comunicação e do conseqüente aumento do número de invasores que tentam burlá-las para tirar proveito financeiro ou obter fama, causando prejuízos às instituições invadidas. Por isso, a segurança de redes, tanto das tradicionais (*wired*) quanto das modernas *wireless* merecem um enfoque especial.

Neste capítulo, serão abordados inicialmente alguns protocolos de segurança genéricos existentes no mercado e que podem ser utilizados tanto para redes *wired* quanto *wireless*. Em seguida, são apresentados os mecanismos de segurança existentes para WLANs IEEE 802.11.

3.1. Protocolos de Segurança

Os protocolos de segurança apresentados a seguir são baseados em RFCs ou *drafts* e podem ser encontrados em produtos do mercado. Por tratarem-se de protocolos gerais, não estão vinculados a utilização de um tipo específico de rede, e dependendo do tipo de aplicação e de possíveis adaptações, podem ser utilizados tanto em redes *wired* quanto *wireless*.

3.1.1. EAP (Extensible Authentication Protocol)

O EAP é um protocolo geral para autenticação que suporta múltiplos mecanismos de autenticação. A definição do protocolo está presente na RFC 2284 [7] e atualizações em sua

estrutura original foram incluídas no *draft* 2284bis [8]. Conforme [8], o EAP foi inicialmente projetado para ser utilizado com conexões discadas PPP, posteriormente adaptado para o uso em redes *wire*d IEEE 802 e tem sido proposto também para redes *wireless*.

O EAP é utilizado para selecionar um mecanismo específico de autenticação, tipicamente após o autenticador requisitar mais informações para determinar os mecanismos específicos de autenticação a serem utilizados. Ao invés de requisitar que o autenticador seja atualizado para suportar cada novo método de autenticação, o EAP permite o uso de um servidor de autenticação que os implemente, com o autenticador atuando como intermediário entre os métodos e os usuários. A descrição aqui apresentada está baseada em [8].

No EAP, são definidos dois elementos básicos: *autenticador* e *peer*. Em [8], é apresentado mais um elemento denominado *servidor de autenticação*. O *autenticador* é o fim do link requerendo a autenticação. O *peer* é o outro fim do link ponto-a-ponto (PPP). A descrição do *peer* em [8], acrescenta que ele também pode ser um segmento de LAN ponto-a-ponto (IEEE 802 *wired media*) ou 802.11 *wireless* link, que está sendo autenticado pelo *autenticador*.

Conforme apresentado em [8], *servidor de autenticação* é uma entidade que fornece um serviço de autenticação para um *autenticador*. Este serviço verifica a identidade do *peer* a partir das credenciais por ele fornecidas. Em [7] não é apresentada a definição de *servidor de autenticação*, por abordar apenas PPP e considerar apenas a presença do *peer* e do *autenticador*.

Funcionamento

De acordo com [7], o funcionamento básico do EAP é o seguinte:

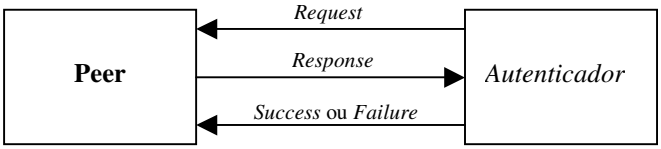

Figura 3.1 - Funcionamento do EAP

1. O *autenticador* envia um pacote *Request* para autenticar o *peer*. O campo tipo deste *Request* indica o que está sendo solicitado, como por exemplo, *Identity*,

MD5-challenge, etc. Tipicamente, o *autenticador* envia um *Identity Request* inicial seguida por outros *Requests* para informações de autenticação.

2. O *peer* envia um pacote *Response* em resposta a cada *Reques*t. Assim como o pacote *Request*, o *Response* contém um campo tipo que corresponde ao campo tipo do *Request*.

3. O *autenticador* finaliza a fase de autenticação com um pacote *Success* ou *Failure*.

A presença de um pacote *Success* no processo de autenticação EAP indica que ele foi concluído com sucesso. Caso contrário, se for gerado um pacote *Failure*, indicará que ocorreu falha na autenticação e conseqüentemente o *peer* não foi autenticado.

Formato dos Pacotes

Os pacotes EAP têm o seguinte formato:

Code 8 bits	**Identifier** 8 bits	**Length** 16 bits	**Data** variável

- *Code*: este campo identifica o tipo do pacote EAP, podendo ser: *Request* (1), *Response* (2), *Success* (3), *Failure* (4).
- *Identifier*: serve para vincular *Requests* com *Responses*.
- *Length*: indica o comprimento do pacote EAP, incluindo os campos *Code*, *Identifier*, *Length* e *Data*.
- *Data*: o seu formato é determinado pelo campo *Code*.

Request e Response

O pacote *Request* é enviado pelo autenticador para o *peer*. O campo *Type* (sub-divisão do campo *Data*) indica o que o *Request* está solicitando. Um *Request* é indicado pelo campo *Code* do pacote EAP igual a 1. *Requests* retransmitidos devem ser enviados com o mesmo *Identifier* para que possam ser distintos de novos *Requests*. O conteúdo do campo *Data* depende do tipo do *Request*.

Após receber um pacote EAP do tipo *Request*, o *peer* deve responder com um pacote *Response* (campo *Code* igual a 2). *Responses* somente devem ser enviados em resposta a *Requests* recebidos e nunca devem ser retransmitidos. O campo *Identifier* de um *Response* deve ser igual ao de seu respectivo *Request*.

Success e Failure

O pacote *Success* é enviado ao *peer* pelo autenticador para confirmar o sucesso da autenticação. O *Success* é um pacote EAP com o campo *Code* igual a 3.

Quando o autenticador não consegue autenticar o *peer* (após receber *Responses* inaceitáveis para um ou mais *Requests*), ele deve enviar um *Failure*, que é um pacote EAP com o campo *Code* igual a 4. Um autenticador pode enviar vários *Requests* antes de gerar um pacote *Failure* a fim de considerar a possibilidade de erros humanos de digitação, por exemplo.

Os pacotes *Success* e *Failure* possuem campo *Data* de tamanho zero e os seus campos *Identifier* devem conter o mesmo valor do *Identifier* do pacote *Response* a que se referem.

Initial EAP Request/Response Types

São os tipos de EAP inicialmente definidos para comunicações *Request/Response*. O campo *Type* tem o tamanho de um byte e identifica a estrutura dos pacotes EAP *Request* ou *Response*. O tipo *Nak* é valido somente para pacotes *Response* e nunca pode ser enviado em um *Request*. Os tipos 1-4 devem ser suportados por todas as implementações EAP. RFCs posteriores poderão definir outros tipos de EAP, adicionalmente aos listados a seguir:

- **Identity**: Este tipo é utilizado para solicitar/responder a identidade do *peer*. Geralmente, o autenticador envia este tipo de pacote como o *Request* inicial, podendo conter uma mensagem a ser exibida no *peer* quando houver interação com o usuário. O pacote *Response* enviado em resposta a um *Request* do tipo *Identity* deve ser do tipo 1 (*Identity*).

- **Notification**: Este tipo é opcionalmente utilizado para transportar uma mensagem do autenticador a ser exibida no *peer*, podendo o último armazena-la em um *log*

caso não possa exibi-la ao usuário. É utilizado para uma notificação de natureza imperativa, como por exemplo, uma senha que esteja próxima de expirar, um aviso de falha de autenticação, etc. Em muitas circunstâncias, notificação não é necessária. O pacote *Response* enviado em resposta a um *Request* do tipo *Notification* deve do tipo 2 (*Notification*).

- **Nak**: Este tipo é válido somente para pacotes do tipo *Response*. É enviado em resposta a um *Request* que contenha uma forma de autenticação inaceitável. As formas de autenticação são indicadas pelos tipos de *Request/Response* (a partir de 4). O pacote *Response Nak* (tipo 3) contém o tipo de autenticação desejado pelo *peer*.

- **MD5-Challenge**: Neste tipo, o *Request* contém uma mensagem "*challenge*" para o *peer*. Um pacote *Response* do tipo 4 (*MD5-Challenge*) ou tipo 3 (*Nak*) deve ser enviado em resposta ao *Request*. Uma resposta *Nak* indica o tipo do mecanismo de autenticação desejado pelo *peer*.

- **One-Time Password (OTP)**: Neste tipo, o *Request* contém uma mensagem a ser exibida incluindo o OTP *challenge*. Um pacote *Response* do tipo 5 (OTP) ou tipo 3 (*Nak*) deve ser enviado em resposta ao *Request*. Uma resposta *Nak* indica o tipo do mecanismo de autenticação desejado pelo *peer*.

- **Generic Token Card**: Este tipo é definido para ser utilizado em várias implementações de *Token Card* que necessitem de input do usuário. O pacote *Request* contém mensagem em texto ASCII e a resposta (*Response*) contêm as informações do *Token Card* necessárias para a autenticação.

3.1.2. TLS (Transport Layer Security)

O protocolo TLS (definido pela RFC 2246 [17]) fornece privacidade de comunicação, permitindo que aplicações cliente/servidor se comuniquem de uma forma que evite

eavesdropping (escuta), *tampering* (alterações) ou *message forgery* (falsificação de mensagem).

Os objetivos principais do protocolo TLS são fornecer privacidade e integridade de dados entre duas aplicações comunicantes. O TLS é composto por duas camadas: *TLS Record Protocol* e *TLS Handshake Protocol*. O nível mais baixo é representado pelo *TLS Record Protocol*, posicionado sobre algum protocolo confiável de transporte (como o TCP). A segurança de conexão fornecida pelo *TLS Record Protocol* tem duas propriedades principais [17]:

- Privacidade de conexão: utiliza criptografia simétrica para cifragem de dados, com as chaves geradas para cada conexão baseadas no segredo negociado pelo *TLS Handshake Protocol*.
- Confiabilidade da conexão: utiliza *message integrity check* para garantir a integridade dos dados.

O *TLS Record Protocol* é usado para encapsular vários protocolos de níveis superiores. Um destes protocolos é o *TLS Handshake Protocol*, que permite a autenticação mútua entre cliente e servidor e a negociação de algoritmo de criptografia e chaves criptográficas antes do protocolo de aplicação transmitir ou receber dados. A segurança de conexão fornecida pelo *TLS Handshake Protocol* tem três propriedades básicas [17]:

- A autenticação da identidade do *peer* pode ser feita usando criptografia assimétrica ou de chave pública.
- Negociação segura do *shared secret* (segredo compartilhado): o segredo negociado fica indisponível para *eavesdroppers* e nas conexões autenticadas o segredo não pode ser obtido, mesmo que um invasor coloque-se no meio da conexão.
- Negociação confiável: o invasor não pode modificar a comunicação de negociação sem ser detectado.

Uma das vantagens do TLS é a sua independência de protocolo de aplicação, o que permite que protocolos de camadas superiores possam ser colocados sobre ele de forma transparente.

Metas

As principais metas do protocolo TLS, em ordem de prioridade, são [17]:

1. Segurança criptográfica: o TLS pode ser utilizado para estabelecer uma conexão segura entre duas partes.
2. Interoperabilidade: programadores independentes podem desenvolver aplicações utilizando TLS que necessitem tocar parâmetros criptográficos sem precisar ter conhecimento dos códigos uns dos outros.
3. Extensibilidade: o TLS procura fornecer um *framework* dentro do qual novos métodos de chave pública e criptografia possam ser incorporados quando necessário.
4. Eficiência relativa: como as operações criptográficas tendem a utilizar intensamente a CPU (principalmente operações de chave pública), o TLS incorporou um esquema opcional de *caching* para reduzir o número de conexões que precisam ser estabelecidas.

TLS Record Protocol

O *TLS Record Protocol* é um protocolo em camadas. Em cada camada, mensagens podem incluir campos para tamanho, descrição e conteúdo. O Record Protocol toma as mensagens, fragmenta os dados, opcionalmente os comprime, aplica o MAC (*Message Authentication Code*, algoritmo utilizado para autenticação da mensagem, garantindo a sua integridade), encripta-os e transmite o resultado. Os dados recebidos sofrem o processo inverso antes de serem entregues aos clientes das camadas superiores.

Existem quatro clientes do *Record Protocol*: *handshake protocol* (gera parâmetros de segurança para a comunicação), *alert protocol* (indica falhas na comunicação), *change cipher spec protocol* (indica transições em estratégias de cifragem) e *application data protocol*. Tipos adicionais podem ser suportados pelo *Record protocol* a fim de possibilitar a extensão do TLS. Se uma implementação TLS receber um *Record type* que não reconheça deve ignorá-lo.

Handshake Protocol

O *TLS Handshake Protocol* é o responsável por gerar os parâmetros criptográficos da sessão, operando sobre a camada *TLS Record*. Isto inclui um acordo entre cliente e servidor sobre: versão do protocolo, algoritmo criptográfico selecionado, autenticação mútua, utilização de técnicas de criptografia de chave pública para gerar *shared secrets* (segredos compartilhados).

O *TLS Handshake Protocol* envolve os seguintes passos [17]:

- Troca de *hello messages* para definir acordo sobre algoritmos, troca de valores aleatórios e checagem para retomada de sessão.
- Troca de parâmetros criptográficos para permitir a cliente e servidor acordo sobre o *premaster secret*.
- Troca de certificados e informações criptográficas para permitir a cliente e servidor autenticar-se mutuamente.
- Gerar a *master secret* a partir da *premaster secret* e valores aleatórios trocados.
- Fornecer parâmetros de segurança para a camada *Record*.
- Permitir a cliente e servidor verificar que seu *peer* calculou os mesmos parâmetros de segurança e que o *handshake* ocorreu sem violação.

Portanto, os parâmetros de segurança do TLS (definição de chaves e algoritmos criptográficos, troca de certificados) ocorrem no *TLS Handshake Protocol*, possibilitando que a posterior comunicação de dados entre cliente e servidor ocorra de maneira segura.

3.1.3. EAP-TLS

O EAP (*Extensible Authentication Protocol*) fornece um mecanismo padrão para suporte a métodos adicionais de autenticação, como *smart cards*, *Kerberos*, *Public Key*, *One Time Passwords*.

Entretanto, vários destes métodos EAP estão focados somente na autenticação do cliente para o servidor, e atualmente tornou-se desejável em muitos casos o suporte a

autenticação mútua, bem como a utilização de um mecanismo de estabelecimento de chaves de sessão. Para suprir estas novas necessidades, o EAP passou a utilizar as funcionalidades proporcionadas pelo protocolo TLS (negociação protegida por cifradores, autenticação mútua, gerenciamento de chaves), constituindo assim o protocolo EAP-TLS [2]. Portanto, o EAP-TLS utiliza-se das características do EAP acrescidas da segurança forte proporcionada pelo TLS.

Visão Geral da Conversação EAP-TLS

O funcionamento da conversação EAP-TLS ocorre conforme representado na figura a seguir.

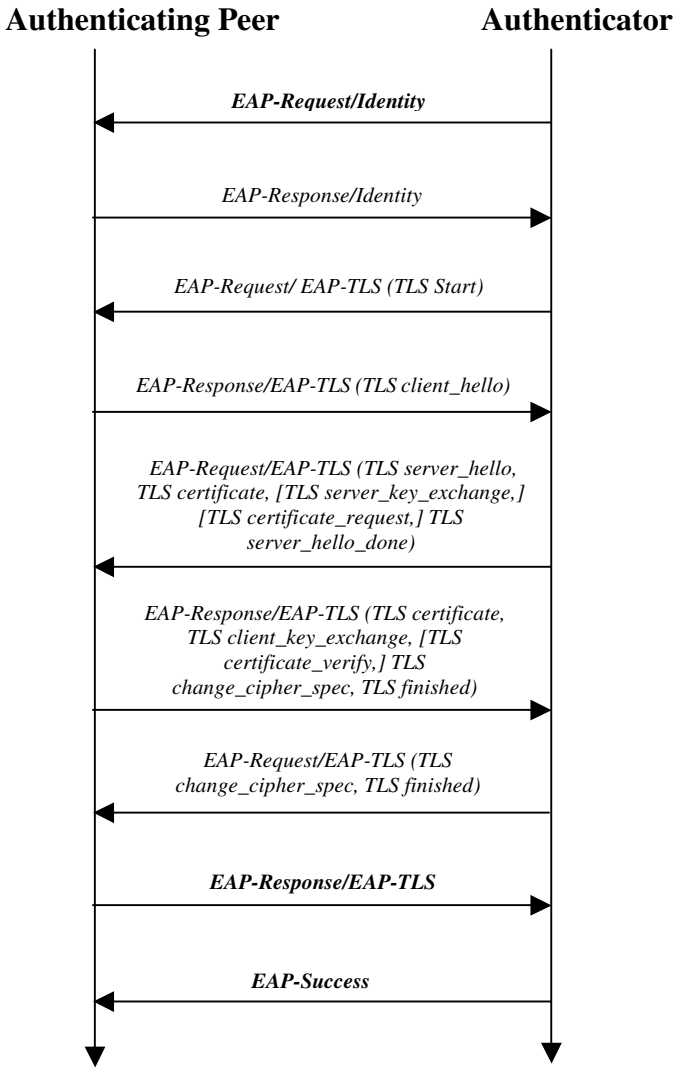

Figura 3.2 - Esquema de funcionamento do EAP-TLS (RFC 2716)

A conversação inicia-se com o autenticador enviando um pacote `EAP-Request/Identity` para o *peer* e este respondendo ao autenticador com um pacote `EAP-Response/Identity` contendo seu userID.

A partir de então, o autenticador torna-se um *passthrough device* (dispositivo intermediário), encapsulando os pacotes EAP recebidos do *pee*r para serem transmitidos ao *backend security server,* possivelmente um servidor RADIUS [33]. Na descrição de funcionamento abaixo, o termo "servidor EAP" é utilizado para referenciar-se ao último ponto de comunicação com o *peer*.

Após receber a identidade do *peer*, o servidor EAP deve responder com um pacote `EAP-Request/EAP-Type=EAP-TLS(TLS Start)` e sem dados. A conversação EAP-TLS inicia-se com o envio de um pacote `EAP-Response/EAP-Type=EAP-TLS(TLS client_hello)` pelo *peer*. O `cipher_spec` neste momento é TLS_NULL_WITH_NULL_NULL e *null compression*, permanecendo o mesmo até que a mensagem `change_cipher_spec` indique uma negociação de atributos.

A mensagem `client_hello` contém: número da versão TLS do cliente (deve ser TLS v1.0 ou posterior), *sessionId*, número aleatório gerado e conjunto de cifradores suportados pelo cliente.

O servidor EAP então responde com um pacote `EAP-Request/EAP-Type=EAP-TLS(TLS server_hello)`, e possivelmente `TLS certificate`, `TLS server_key_exchange`, `TLS certificate_request`, `TLS server_hello_done` e/ou `TLS finished`, e/ou `TLS change_cipher_spec`. A mensagem `TLS server_hello` contém: número da versão do TLS (v.1.0 ou posterior), número aleatório gerado pelo servidor, *sessionId* e cifrador).

Caso o *sessionId* do cliente seja nulo ou irreconhecível pelo servidor, este deve escolher um *sessionId* para estabelecer uma nova sessão. O servidor também deve escolher um cifrador entre os oferecidos pelo cliente. Se o cliente fornecer um *sessionId* conhecido pelo servidor, indicando a restauração de uma sessão previamente estabelecida, então o cifrador deve ser o mesmo negociado durante o *handshake protocol* no estabelecimento da sessão.

O propósito da utilização do *sessionId* com TLS é melhorar a eficiência nos casos em que o cliente esteja tentando autenticar-se com o servidor EAP dentro de um curto período de tempo.

A decisão de tentar retomar uma sessão anterior para encurtar a conversação TLS é feita pelo cliente, baseada principalmente no tempo transcorrido desde a tentativa anterior de autenticação. Entretanto, a decisão de continuar ou definir uma nova sessão é tomada pelo servidor EAP, baseada no *sessionId* fornecido pelo *peer* e no tempo transcorrido desde a autenticação anterior.

Nos casos em que o autenticador e o servidor EAP estão no mesmo dispositivo, o cliente poderá continuar sessões somente quando conectar-se ao mesmo NAS (*Network Authentication Server*). Quando a autenticação EAP é remota (autenticador e servidor EAP localizados em dispositivos diferentes) a continuidade de sessões é mais viável, pois múltiplos NAS podem utilizar o mesmo servidor RADIUS para autenticação.

Quando o servidor EAP retomar uma sessão previamente estabelecida, deve incluir somente as mensagens `TLS change_cipher_spec` e `TLS finished` (contendo a resposta de autenticação do servidor EAP para o *peer*) após a `TLS server_hello`. Se o servidor EAP não estiver retomando uma sessão previamente estabelecida, então deve incluir a `TLS server_certificate`, e `TLS server_hello_done` deve ser a última mensagem encapsulada no pacote `EAP-Request`.

A mensagem `certificate` contém cadeia de certificados de chave pública tanto para troca de chave pública quanto para assinatura de chave. Para possibilitar a troca de chave, a mensagem `TLS server_key_exchange` deve ser incluída.

A mensagem `TLS certificate_request` é utilizada quando o servidor deseja que o cliente se autentique via chave pública.

O *peer* deve responder ao `EAP-Request` com um pacote `EAP-Response/EAP-Type=EAP-TLS(TLS change_cipher_spec, TLS finished)` e possivelmente (`TLS certificate`, `TLS certificate_verify` e/ou `TLS client_key_exchange`). Se a mensagem `TLS server_hello` contida no `EAP-Request` anterior indicar a retomada de uma sessão prévia, então o *peer* deve enviar somente `TLS change_cipher_spec` e `TLS finished` (contém a resposta de autenticação do *peer* para o servidor EAP).

Se a mensagem `TLS server_hello` anterior não indicar a retomada de uma sessão prévia, o *peer* deve enviar adicionalmente `TLS client_key_exchange`, para completar a troca do *shared master secret* (segredo compartilhado) entre o *peer* e o servidor EAP. Se o servidor EAP enviou uma mensagem `TLS certificate_request` no pacote `EAP-`

Request anterior, o *peer* deve enviar também TLS certificate e TLS certificate_verify. A primeira contém um certificado para a assinatura de chave pública do *peer* e o segundo a resposta de autenticação assinada pelo *peer* para o servidor EAP. Após receber o pacote, o servidor EAP verifica o certificado e assinatura digital do *peer*.

Se o *peer* não for autenticado, o servidor EAP precisa enviar um pacote EAP-Request/EAP-Type=EAP-TLS(TLS alert) para permitir ao *peer* informar ao usuário a causa da falha de autenticação.

Para garantir que o *peer* recebeu a mensagem TLS alert, o servidor EAP deve aguardar a sua resposta. Se o pacote EAP-Response recebido contiver a mensagem TLS client_hello, o servidor EAP pode permitir que a conversação seja reiniciada. Se o pacote EAP-Response não contiver dados, o servidor EAP deve enviar um pacote EAP-Failure e finalizar a conversação. As decisões de quando permitir reinícios e de quantas vezes a conversação pode ser reiniciada ficam a cargo do servidor EAP.

Se o *peer* for autenticado, o servidor EAP deve responder com pacote EAP-Request/EAP-Type=EAP-TLS(TLS change_cipher_spec, TLS finished) caso seja uma nova sessão TLS. A mensagem TLS finished contém a resposta de autenticação do servidor EAP para o *peer*. O *peer* deve verificar o *hash* para autenticar o servidor EAP.

Se o servidor EAP não for autenticado, o *peer* precisa enviar um pacote EAP-Response/EAP-Type=EAP-TLS(TLS Alert) identificando a razão da falha na autenticação do servidor.

Para assegurar que o servidor EAP recebeu a mensagem TLS Alert, o *peer* deve esperar a resposta do servidor. No caso de falha de autenticação do servidor, o servidor EAP deve responder com um pacote EAP-Failure, pois falha na autenticação do servidor é uma condição terminal.

Se o servidor EAP for autenticado, o *peer* deve enviar um pacote EAP-Response/EAP-Type=EAP-TLS sem dados. O servidor EAP deve então responder com um pacote EAP-Success.

Formato do pacote EAP-TLS

A estrutura de um pacote EAP-TLS é a seguinte:

Code 8 bits	**Identifier** 8 bits	**Length** 16 bits	**Type** 8 bits	**Data** variável

- *Code*: *Request* (1), *Response* (2).
- *Identifier*: vincula *responses* a *requests*. Este campo deve ser alterado em cada pacote do tipo *Request*.
- *Length*: indica o tamanho do pacote EAP, incluindo os campos *Code*, *Identifier*, *Length*, *Type* e *Data*.
- *Type*: 13 – EAP TLS.
- *Data*: o formato do campo *Data* é determinado pelo campo *Code*.

No caso dos pacotes *Request* e *Response* o campo *Data* subdivide-se em:

- *Flags*: o bit L assinalado indica a presença de quatro bytes no campo *TLS Message Length* e deve estar assinalado no primeiro fragmento de uma mensagem TLS fragmentada ou conjunto de mensagens. O bit M é assinalado em todos menos no último fragmento. O bit S é assinalado na mensagem *EAP-TLS Start*.
- *TLS Message Length*: tem o tamanho de quatro bytes e está presente somente se o bit L estiver assinalado. Fornece o tamanho total da mensagem TLS ou conjunto de mensagens que está sendo fragmentado.
- *TLS data*: consiste do pacote TLS encapsulado no formato *TLS record*.

Aplicação

Uma forma de aplicação do EAP-TLS em ambiente WLAN, especificamente IEEE 802.11, é apresentada pela Cisco em [12]. Nesta utilização, existem algumas diferenças em relação ao inicialmente descrito em [2]:

- Foco em redes *wireless*, aplicando a descrição conceitual apresentada em [2].

- Presença de um servidor RADIUS utilizado como servidor de autenticação.
- Utilização de um *Access Point* como elemento intermediário entre o *Authenticating Peer* e o servidor de autenticação. Apresenta um AP proprietário (*Cisco Aironet*).

Este exemplo de utilização apresentado em [12] demonstra a viabilidade de utilização do EAP-TLS como forma de aumentar a segurança em redes *wireless*, e neste caso específico, redes IEEE 802.11.

3.1.4. Outras variantes e extensões do EAP

Além do EAP-TLS definido em RFC e provavelmente o tipo mais utilizado e um dos que proporciona uma das melhores infra-estruturas de segurança entre os EAPs, existem ainda outros tipos, dentre os quais destacam-se: LEAP, PEAP e EAP-TTLS.

LEAP (EAP Cisco Wireless)

Trata-se de um tipo de autenticação para WLANs (*wireless* LANs) que suporta autenticação mútua entre o cliente e o servidor RADIUS através de senhas, além de fornecer chaves WEP dinâmicas por usuário e por sessão. Foi criado pela Cisco em dezembro de 2000.

De acordo com [16], o funcionamento do LEAP em uma WLAN é o seguinte:

- O cliente *wireless* associa-se com o *access point* (AP).
- O AP bloqueia qualquer tentativa de acesso à rede pelo cliente até que ele tenha sucesso na fase de autenticação.
- O usuário do cliente fornece *username* e senha.
- Utilizando 802.1X e EAP, o cliente *wireless* e o servidor RADIUS realizam autenticação mútua através do AP. O servidor RADIUS envia um *challenge* de autenticação para o cliente. O cliente utiliza o *hash* da senha do usuário para elaborar uma resposta ao *challenge* e a envia para o servidor RADIUS. A partir de informações da sua base de dados de usuários, o servidor RADIUS cria a sua resposta e compara-a com a recebida do cliente. Após autenticar o cliente, o

mesmo processo ocorre em sentido inverso para que o cliente autentique o servidor.

- Após a conclusão com sucesso da autenticação mútua, o servidor RADIUS e o cliente determinam a chave WEP que será utilizada pelo cliente durante o *logon*.
- O servidor RADIUS envia a WEP *key* (chave de sessão) para o AP.
- O AP criptografa a sua chave de *broadcast* com a chave de sessão e a envia para o cliente.
- O cliente e o AP ativam o WEP e utilizam as chaves WEP de sessão e *broadcast* para a comunicação durante a validade da sessão.
- As duas chaves são trocadas em intervalos definidos pelo servidor RADIUS.

Segundo [11], o LEAP suporta como sistemas operacionais clientes as plataformas Windows (XP, 2000, Me, NT, 98 e 95), Windows CE, Linux, Mac OS e MS-DOS.

PEAP (Protected EAP)

Trata-se de um tipo de autenticação projetado para aproveitar as vantagens do EAP-TLS, autenticando o servidor através de certificados, e suportar vários métodos para a autenticação do cliente, incluindo senhas de *logon* e OTPs (*One-Time Passwords*).

Além de utilizar a autenticação do servidor por certificados, o PEAP também aproveita a segurança proporcionada pelo EAP-TLS para a comunicação entre cliente e servidor, inclusive protegendo a autenticação do cliente dentro do TLS [35].

De acordo com [11], o funcionamento do PEAP está dividido em duas fases:

- Na fase 1, a autenticação TLS do lado servidor é realizada para criar um túnel criptografado e realizar a autenticação de maneira similar a autenticação de um servidor Web utilizando SSL (*Secure Sockets Layer*). Após estabelecida a fase 1 do PEAP, todos os dados são criptografados, incluindo as informações sensíveis do usuário.

- Na fase 2, o PEAP possibilita que a autenticação do cliente seja feita utilizando vários métodos diferentes, como por exemplo OTP.

Vale a pena salientar que o PEAP ainda não é um padrão. Ele baseia-se em um I-D (*Internet Draft*) enviado à IETF por *Cisco Systems*, *Microsoft* e *RSA Security*. De acordo com [11], suporte ao PEAP está disponível no sistema operacional *Microsoft Windows XP*.

EAP-TTLS (EAP Tunneled TLS)

Trata-se de um tipo de autenticação (similar ao PEAP) projetado para aproveitar as vantagens do TLS, autenticando o servidor através de certificados, e fornecer um canal seguro para métodos de autenticação PPP tradicionais como CHAP, MS-CHAP, MS-CHAP-V2 e EAP/MD5, utilizados do lado cliente. Foi proposto pela empresa *Funk Software*.

De acordo com [35], talvez a característica mais atrativa do EAP-TTLS seja a capacidade de continuar utilizando servidores RADIUS legados para autenticar *wireless* LANs através da inserção de um servidor RADIUS/EAP-TTLS entre os *wireless* APs e o servidor RADIUS legado.

Como ponto negativo do EAP-TTLS pode-se citar a utilização de métodos de autenticação menos seguros do lado cliente, que mesmo protegidos dentro do TLS tornam-se mais suscetíveis a ataques que a utilização de certificados, por exemplo.

3.2. Segurança Disponível para Redes IEEE 802.11

Inicialmente, a segurança proporcionada por WLANs padrão IEEE 802.11 se resumia à utilização de dois mecanismos rudimentares de controle de acesso (SSID e filtragem de endereços MAC) e um mecanismo de privacidade (WEP), sendo algumas vezes utilizados em conjunto para tentar fornecer um nível maior de segurança.

Com o passar do tempo, foram desenvolvidas outras opções para melhorar a proteção em redes *wireless*, como a criação do padrão IEEE 802.1X e a incorporação de mecanismos de camadas superiores, como por exemplo EAP e TLS, à infra-estrutura de segurança das redes IEEE 802.11.

3.2.1. Mecanismos de controle de acesso

Os mecanismos de controle de acesso surgiram como tentativa de barrar invasores às redes IEEE 802.11 quando estas foram criadas, sendo também utilizados em conjunto com o WEP para tentar garantir uma maior segurança.

SSID (Service Set Identifier)

Trata-se de um mecanismo que permite a WLANs serem segmentadas em múltiplas redes, cada uma com um identificador diferente. Por exemplo, um edifício pode ser segmentado em múltiplas redes por andar ou departamento. A cada uma destas redes é atribuído um identificador único programado em um ou mais APs (uma rede pode consistir de múltiplos APs). Para acessar uma dessas redes, o computador cliente deve estar configurado com o identificador SSID correspondente. Portanto, pode-se considerar que o SSID atua como uma forma de senha para controle de acesso.

Como principal vulnerabilidade deste mecanismo, pode-se citar o fato de o SSID ser transmitido na forma de texto desprotegido em mensagens enviadas pelo *access point*, o que possibilita que um *eavesdropper* possa facilmente determinar o SSID utilizando um analisador de pacotes 802.11. Alguns fabricantes de *access points* oferecem a opção de desabilitar o *broadcast* do SSID nas mensagens, mas isto pode causar problemas de interoperabilidade. De acordo com [34], o SSID não foi criado para servir como um mecanismo de segurança.

Filtragem de endereços MAC

É uma forma de controle de acesso suportada por produtos de alguns fabricantes, baseada no endereço físico (endereço MAC) do NIC (*Network Interface Card*) do cliente. O *access point* permite associação somente se o endereço MAC do cliente estiver presente na tabela de controle de acesso utilizada pelo AP.

Entre as principais vulnerabilidades deste mecanismo, pode-se citar o fato de os endereços MAC serem enviados sem proteção, como requerido pela especificação 802.11. Como resultado, em LANs *wireless* que utilizam filtragem de endereços MAC, o invasor da rede pode ser capaz de subverter o processo por *spoofing* de um endereço MAC válido [34].

Além da possibilidade de um endereço MAC ser forjado, existe a possibilidade do NIC poder ser perdido ou roubado [11].

3.2.2. Padrão IEEE 802.11

A segurança definida no padrão IEEE 802.11 envolve autenticação e privacidade no nível da camada de enlace e, portanto, aplicações que necessitem de um nível de segurança maior devem complementá-la com mecanismos de segurança proporcionados pelas camadas superiores.

Quanto à autenticação, ela pode ser de dois tipos: *Open System* ou *Shared Key*. Eles podem ser utilizados tanto para autenticação entre estação e AP (*infrastructure BSS*) quanto para autenticação entre estações em rede *ad-hoc* (IBSS).

Como forma de garantir a privacidade dos dados, o padrão IEEE 802.11 utiliza o WEP (*Wired Equivalent Privacy*).

Autenticação *Open System*

É a forma de autenticação mais simples, sendo também chamada de autenticação nula. Qualquer STA (estação) que solicite autenticação com este algoritmo pode tornar-se autenticada. Autenticação do tipo *Open System* não é necessariamente concluída com sucesso, pois uma STA pode recusar a autenticar-se com uma outra STA em particular. É o algoritmo de autenticação *default* para redes IEEE 802.11.

Esta forma de autenticação envolve uma seqüência de dois passos:

1. A estação solicitante declara a sua identidade e solicita autenticação.
2. A estação solicitada informa o resultado da autenticação.

Se o resultado da autenticação for "*successful*", as STAs estão mutuamente autenticadas.

Autenticação *Shared Key*

Este tipo suporta autenticação de STAs que possuam o conhecimento de uma chave secreta compartilhada. A autenticação por *Shared Key* ocorre sem a necessidade de transmitir a chave secreta de forma aberta (sem proteção), mas exige o uso do mecanismo de privacidade WEP (*Wired Equivalent Privacy*).

A chave secreta deve ser entregue às STAs participantes através de um canal seguro que é independente do IEEE 802.11.

Dentro do tipo *Shared Key*, a STA que inicia a troca de autenticação é referenciada como *requester* e a que recebe o primeiro frame desta troca é chamada *responder* [28]. Esta forma de autenticação envolve uma seqüência de quatro passos:

1. O *requester* envia um frame *authentication request* ao *responder* (AP) solicitando autenticação por *shared key*.
2. O *responder* responde com um frame *authentication response* contendo um *challenge*.
3. O *requester* cifra o *challenge* com sua chave WEP e o envia em um novo *authentication request*.
4. Se o *responder* conseguir decifrar o *authentication request* e obter o *challenge* original, ele responde com um *authentication response* concedendo acesso ao *requester*.

Vale salientar que para realizar a autenticação por *Shared Key* é necessário que a chave WEP já esteja armazenada nas estações comunicantes.

WEP (*Wired Equivalent Privacy*)

O mecanismo de privacidade denominado WEP foi definido pelo padrão IEEE 802.11 para proteger usuários autorizados da *wireless* LAN de *eavesdropping* casual. O seu objetivo é fornecer funcionalidade para *wireless* LAN equivalente a fornecida pelos atributos de segurança física inerentes ao meio *wired* [28].

A privacidade de dados oferecida pelo WEP baseia-se em chaves criptográficas simétricas de 40 bits e no IV (*Initialization Vector*) público de 24 bits.

Funcionamento

O WEP utiliza-se de uma chave secreta *k* compartilhada entre as partes comunicantes a fim de proteger o corpo do *frame* de dados. A cifragem ocorre da seguinte forma:

- *Checksumming*: primeiro é computado um *intergrity checksum* `c(M)` a partir da mensagem *M*. Depois, é concatenado o *checksum* com a mensagem a fim de obter o *plaintext* `P = (M, c(M))`, que será utilizado como entrada para o estágio seguinte.

- *Cifragem*: neste estágio, o texto aberto *P* é criptografado utilizando RC4. Deve ser escolhido um vetor de inicialização (IV) *v*. O algoritmo então gera uma *keystream* (seqüência de bytes pseudoaleatórios) em função do IV *v* escolhido e da chave *k*, representado por `RC4(v,k)`. Então, é realizado um *xor* (ou-exclusivo) entre o texto aberto e a *keystream* para obter o texto cifrado.

- *Transmissão*: finalmente, o IV e o texto cifrado são transmitidos através do link de rádio.

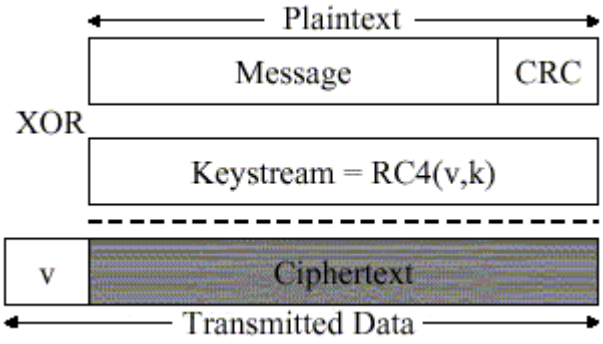

Figura 3.3 -Encrypted WEP Frame (obtido de [9])

Para decifrar um *frame* protegido por WEP, o receptor deve realizar o processo inverso da cifragem. Primeiro é re-gerada a *keystream* RC4 (*v, k*) e realizado um *xor* entre ela e o texto cifrado para recuperar o texto aberto inicial. Em seguida, o receptor deve verificar o

checksum do texto aberto decriptado *P'* particionando-o na forma (M', c'), re-computando o *checksum* c(M'), e comparando se é igual ao *checksum c* recebido. Isto garante que somente *frames* com *checksum* válido serão aceitos pelo receptor.

Novas alternativas ao WEP

Apesar da tentativa de propor um bom nível de segurança para redes IEEE 802.11 através da utilização do WEP no nível da camada de enlace, em [9, 3, 37] foram identificadas muitas falhas neste protocolo, como por exemplo, repetição do mesmo IV e reuso de chaves (facilita a obtenção dos dados por um invasor), fragilidade no mecanismo de integridade (possibilita que modificações nos dados não sejam percebidas).

Para melhorar a segurança na camada de enlace, a IEEE criou uma solução temporária chamada TKIP (*Temporal Key Integrity Protocol*). O TKIP é um conjunto de algoritmos que adapta o protocolo WEP para suprir algumas de suas falhas conhecidas. De acordo com [37], o TKIP acrescenta os seguintes elementos:

- MIC (*Message Integrity Code*) para evitar fraudes.
- Sequenciamento de pacotes, para evitar *replay attacks*.
- Construção de chaves criptográficas por pacote, para evitar ataques FMS (*Fluhrer-Mantin-Shamir* [21]).

Como solução de longo prazo, a IEEE está desenvolvendo o CCMP (*Counter-Mode-CBC-MAC Protocol*), também com o objetivo de resolver as deficiências do WEP. Este novo protocolo utiliza o algoritmo de criptografia AES (*Advanced Encryption System*) e é incompatível com o WEP.

3.2.3. Padrão IEEE 802.1X

Trata-se do padrão apresentado em [29], que define um mecanismo para controle de acesso a rede, utilizando-se das características físicas de acesso das infra-estruturas IEEE 802 LAN para fornecer meios de autenticação e autorização a dispositivos conectados ponto-a-ponto, e para evitar acesso a ela quando os processos de autenticação e autorização falharem.

Para o contexto específico de WLANs 802.11, o padrão IEEE 802.1X surgiu como uma possibilidade de melhorar a segurança até então existente.

Segundo o padrão [29], os pontos de conexão à LAN podem ser físicos, como por exemplo, um MAC conectado a um segmento físico da LAN, ou lógicos, como uma associação IEEE 802.11 entre uma estação e um AP.

É importante salientar que o foco do padrão IEEE 802.1X são as camadas física e enlace do modelo ISO/OSI. Adicionalmente, define o EAP como meio de comunicar as informações de autenticação entre o solicitante e o servidor de autenticação.

Definições

Os três elementos principais abordados no padrão IEEE 802.1X são definidos em [29] como:

- *Autenticador*: entidade localizada em uma ponta de um segmento ponto-a-ponto da LAN que facilita a autenticação da entidade conectada à outra ponta deste link.

- *Servidor de autenticação*: entidade que fornece serviço de autenticação a um autenticador. Este serviço determina, a partir das credenciais fornecidas pelo solicitante, quando ele está autorizado a acessar os serviços fornecidos pelo autenticador.

- *Solicitante*: entidade localizada em uma das pontas do segmento ponto-a-ponto da LAN que está sendo autenticada pelo autenticador conectado a outra ponta deste link.

Dentre os termos acima descritos, a única nomenclatura nova é a do solicitante, que em aplicações é também chamado por outros nomes como, por exemplo, *peer*, cliente ou STA (estação).

Funcionamento

O funcionamento de uma comunicação dentro do padrão IEEE 802.1X ocorre da seguinte forma:

Inicialmente, o autenticador permite que por ele trafeguem dados independentemente do estado do solicitante (autenticado ou não autenticado), desde que quando não autenticado, os dados estejam endereçados ao servidor de autenticação, pois neste caso, o solicitante não pode ter acesso à rede. Portanto, geralmente os dados iniciais são utilizados para que o solicitante seja autenticado pelo servidor de autenticação e posteriormente possa comunicar-se diretamente com o autenticador e ter acesso à rede.

Resumidamente, até que a autenticação tenha sido concluída com sucesso, o solicitante somente tem acesso ao autenticador para realizar trocas de autenticação, ou para acessar serviços oferecidos por ele que não estejam sujeitos as restrições do controle de acesso. Após a autenticação ser completada com sucesso, o autenticador permite acesso aos serviços de rede por ele oferecidos.

A autenticação nas camadas superiores (acima da camada de enlace), de acordo com [29] deve ser realizada utilizando EAP (*Extensible Authentication Protocol*). Para as comunicações entre o solicitante e o autenticador (no caso do padrão IEEE 802.11 são respectivamente uma estação *wireless* e um *access point*), foi definido que os pacotes EAP devem estar encapsulados dentro do protocolo EAPOL (EAP Over LAN), que também fornece funções de controle como *start* e *logoff*. Como servidor de autenticação, em [29] é sugerida a utilização de servidor RADIUS.

3.3. Conclusão

Este capítulo apresentou protocolos genéricos de segurança, com foco no EAP e sua aplicação com o TLS, gerando o EAP-TLS. Posteriormente, foram descritos os mecanismos de segurança utilizados em WLANs IEEE 802.11 e o padrão IEEE 802.1X que pode ser utilizado tanto em redes 802 *wired* (como *Ethernet*) quanto *wireless* (802.11).

Dentre as opções apresentadas, optou-se por utilizar no trabalho o conjunto de protocolos EAP-TLS por tratarem-se de protocolos não proprietários e de grande aceitação na comunidade internacional; por proporcionarem um alto nível de segurança comparativamente

com outros protocolos como LEAP, PEAP e EAP-TTLS; por possibilitarem a autenticação mútua por certificados; e poderem ser utilizados na complementação da segurança oferecida pelo padrão IEEE 802.11 acima da camada de enlace.

Capítulo 4

SPKI

Em 1996 foi proposta por Lampson e Rivest uma nova infra-estrutura de chave pública denominada SDSI (*Simple Distributed Security Infrastructure*) [15]. Tinha como características principais um espaço de nomes descentralizado, onde o proprietário de cada chave pública poderia criar um espaço de nomes local relativo a esta chave. Estes espaços poderiam ser conectados de uma maneira flexível a fim de possibilitar a criação de cadeias de autorização e definir grupos de *principals* autorizados.

Concorrentemente a SDSI, a SPKI (*Simple Public Key Infrastructure*) [15, 18, 19] estava sendo desenvolvida por Carl Ellison, Bill Frantz, Brian Thomas e Tatu Ylonen, visando criar uma infra-estrutura de chave pública extremamente simples quanto ao design e a flexibilidade em especificar autorizações. Seu funcionamento básico consistia em identificar *keyholders* através de uma chave pública (ou *hash* desta chave) e então vincular autorizações a ela.

Parte da motivação por trás do desenvolvimento tanto da SDSI quanto da SPKI foi a mesma: a complexidade e a falta de flexibilidade da infra-estrutura de chave pública do padrão X.509.

Após a percepção de idéias em comum, em 1997 houve a união de esforços entre SDSI e SPKI, resultando no chamado SPKI/SDSI. Algumas vezes, para abreviar, a união entre elas é referenciada somente por SDSI ou SPKI (forma atualmente mais utilizada).

Ainda na década de 90 foi formado na IETF o grupo de trabalho SPKI que a partir de então se tornou o responsável por continuar o refinamento e evolução do SPKI.

4.1. Nomes SPKI/SDSI

Em SPKI/SDSI utiliza-se um *local name space* (espaço de nomes local) associado a cada chave pública. Não existem nomes globais em SPKI/SDSI (eles existiram somente na primeira versão da SDSI e foram abolidos após a fusão com SPKI). Um nome local é um par composto de uma chave pública e um identificador.

Uma chave privada (relacionada a uma chave pública) pode assinar certificados vinculando um nome local a um valor (nome, chave ou *hash* de chave).

4.2. Chaves

Em SPKI/SDSI todos os *principals* são representados por suas chaves públicas. Um *principal* pode ser um indivíduo, processo ou entidade ativa cujas mensagens possam ser reconhecidas através de sua assinatura digital (produzida por sua chave privada que a representa).

Segundo [15], a simbologia utilizada para representar uma chave pública é composta pela letra *K* acompanhada por um identificador (ex. K_1, K_2, K_A, K_B, *K'*). Quando se afirma que uma mensagem foi assinada pela chave *K'*, significa que ela foi assinada pela chave secreta correspondente a chave pública *K'*.

Apesar de utilizar-se a representação simplificada (*Kidentificador*), uma chave está sempre associada a uma estrutura de dados composta pelo nome do algoritmo utilizado (ex. RSA com *hashing* MD5) e parâmetros associados.

Um identificador é uma palavra gerada a partir de um alfabeto padrão. Por exemplo, *A*, *B*, *Alice*, *Bob*, etc.

4.3. Nomes locais e espaços de nomes locais

Cada chave (pública) possui um espaço de nome local; não existe *global name space* (espaço de nome global) ou uma hierarquia de espaços de nomes. Um nome local é uma seqüência de dois elementos, consistindo de uma chave K_X e um identificador. Exemplos: K_X *Alice*, K_X *project-team* (K_X representa a chave pública do emissor). Um nome local "K_X A" pertence ao espaço de nomes local da chave K_X.

Os nomes locais em diferentes espaços de nomes são independentes entre si, mesmo que utilizem o mesmo identificador. O proprietário da chave pública (emissor do certificado) é quem decide que tipo de identificadores ele deseja utilizar para atribuir nomes no seu espaço local, podendo ser, por exemplo, nomes de pessoas, apelidos, números de telefones, funções organizacionais ou nomes de grupos.

De acordo com [15], algumas vantagens de se utilizar nomes locais são:

- Fornecer uma maneira amigável de referenciar-se a outro *principal*. Por exemplo, é mais fácil referenciar-se a um nome que à sua respectiva chave pública.
- Fornecer um nível de abstração que separe o nome de um *principal* de suas chaves, com o objetivo de facilitar a sua atualização posterior, pois estas chaves podem futuramente mudar.
- Permitir que um *principal* defina um nome em função do nome definido por outro principal.
- Permitir nomes que referenciem um grupo de *principals*, como por exemplo "amigos".
- Permitir nomes de grupos que possam ser utilizados como atributos *booleanos* e atribuídos aos *principals* que os possuem, como por exemplo "maior-de-21-anos".

Além dos nomes locais simples (composto por uma chave e um identificador) existe em SPKI o *extended name* (nome estendido), também conhecido por nome composto, que é uma seqüência formada por uma chave seguida por dois ou mais identificadores.

Exemplos de *extended names*: "K_X Alice mãe", "K_X PUCPR aluno".

4.4. Certificados

Existem dois tipos de certificados (também chamados de *certs*) em SPKI/SDSI: *name certs* (certificados de nomes) que vinculam chaves a nomes locais e *authorization certs* (certificados de autorização) que conferem autorização a um nome ou uma chave.

4.4.1. Certificados de Nomes

Um *name cert* (certificado de nome) é utilizado para vincular a chave pública de uma entidade (sujeito) a um nome local (por exemplo, K_S *Alice*) pertencente ao espaço de nome local do emissor (*issuer*) (por exemplo, K_S). Somente a chave K_S pode assinar certificados para nomes no seu espaço de nome local. Um certificado de nome é uma estrutura assinada pela chave privada do emissor e representada por (I, N, S, V):

- *Issuer* (I): é a chave pública do emissor, que assina o certificado com a respectiva chave privada.
- *Name* (N): é o identificador que juntamente com a chave do emissor define o nome local "K_S Alice", que pertence ao espaço de nomes local da chave K_S.
- *Subject* (S): é o sujeito (identificado por uma chave ou um nome) representado pelo nome local "K_S Alice".
- *Validity dates* (V): é a especificação de validade do certificado na forma (*T1*, *T2*), representando que o certificado é válido do tempo *T1* até o tempo *T2* (inclusive). Se o tempo *T1* (*not-before*) não estiver presente é assumido o valor -∞. Se o tempo *T2* (*not-after*) estiver ausente é assumido o valor +∞.

A seguir está representado um exemplo de certificado de nome:

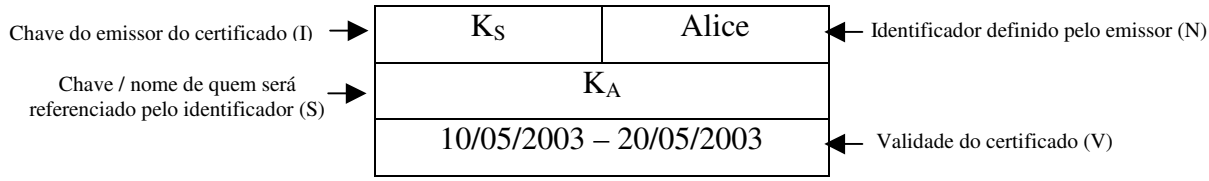

Figura 4.1 - Exemplo de certificado de nome

4.4.2. Certificados de Autorização

A função de um *auth cert* (certificado de autorização) é garantir ou delegar uma autorização específica do emissor (*issuer*) para o sujeito (*subject*).

Um certificado de autorização é uma estrutura da forma (I, S, D, A, V), onde:

- *Issuer* (I): é a chave pública do emissor (*issuer*) que assina o certificado. O emissor é quem concede a autorização.
- *Subject* (S): é um nome ou chave pública que representa o sujeito beneficiário da autorização.
- *Delegation* (D): é o bit de delegação que quando assinalado como verdadeiro garante ao sujeito permissão para delegar a autorização que está recebendo.
- *Authorization* (A): é a especificação de autorização que define as permissões que estão sendo concedidas pelo emissor para o sujeito.
- *Validity dates* (V): é a especificação de validade do certificado na forma (*T1*, *T2*), representando que o certificado é válido do tempo *T1* até o tempo *T2* (inclusive). Se o tempo *T1* (*not-before*) não estiver presente é assumido o valor -∞. Se o tempo *T2* (*not-after*) estiver ausente é assumido o valor +∞.

A seguir está representado um exemplo de certificado de autorização:

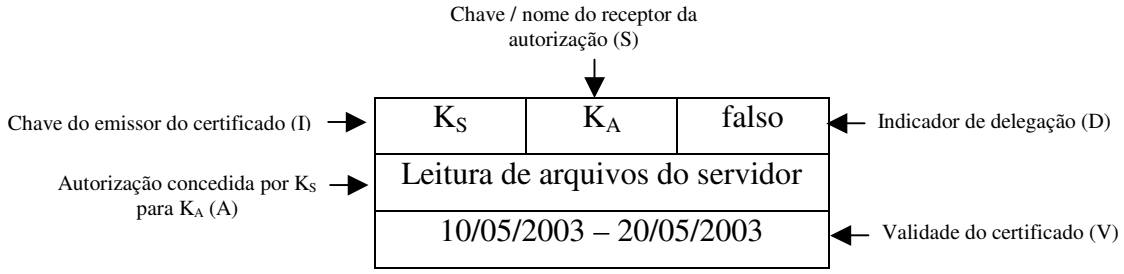

Figura 4.2 - Exemplo de certificado de autorização

É importante salientar que um certificado de autorização não invalida outros, sendo, portanto o seu efeito cumulativo. Como não existem certificados negativos, uma permissão concedida permanece válida até que os certificados envolvidos expirem ou tornem-se inválidos.

Em SPKI/SDSI os certificados de autorização e de nomes se integram da seguinte maneira: enquanto os *name certs* são utilizados para vincular nomes simbólicos às chaves públicas (ou outros nomes), os *auth certs* são responsáveis por autorizar operações a elas.

De acordo com [15], a semântica das requisições e das autorizações é aberta. A sua definição deve ser feita em função da implementação.

4.4.3. Delegação

Uma das principais características da SPKI é a possibilidade de delegação de autorizações presente nos certificados de autorização. O campo delegação é do tipo *booleano* e tem a função de permitir que a autorização concedida por um *principal A* para um *principal B* através de um certificado possa ser repassada total ou parcialmente a um *principal C* (assim como o poder de delegação), desde que o certificado emitido para *B* contenha o bit de delegação com o valor verdadeiro.

Caso um *principal* possua um certificado que lhe conceda uma determinada autorização, mas com o bit de delegação com valor falso, este não poderá delegar a autorização recebida para outro *principal*.

Vale salientar que um *principal* nunca pode delegar uma autorização superior à que possui.

4.4.4. Redução de Certificados de Autorização

De acordo com [36], o processo de redução dos certificados é importante principalmente para a validação e a simplificação de cadeias de certificados. Este processo (também referenciado como *5-tuple reduction*) é utilizado principalmente pelo servidor, após receber os certificados do cliente, para computar os seus direitos. Após verificar a assinatura dos certificados, o servidor pode simplificar um conjunto de certificados para *5-tuples*. Então, o processo de *5-tuple reduction* pode ser utilizado para calcular os direitos do cliente e adicionalmente simplificar um conjunto de certificados para um único.

Segundo [19], *5-tuple* é uma forma intermediária mantida em uma memória confiável para que não precise de uma assinatura digital. É produzida a partir de certificados utilizando um software confiável e seu conteúdo é o mesmo do corpo de um certificado de autorização SPKI. Portanto, a diferença entre eles está na presença de assinatura (certificado) ou na sua ausência (*5-tuple*).

O exemplo de um processo de redução é apresentado a seguir:

A partir de duas *5-tuples*:

<I1, S1, D1, A1, V1> (1)
<I2, S2, D2, A2, V2> (2)

Aplicando redução pode-se gerar a *5-tuple*:

<I1, S2, D2, AIntersect (A1, A2), VIntersect (V1, V2)> (3)

Que é equivalente às anteriores, desde que S1=I2 e D1=verdadeiro.

Posteriormente, a *5-tuple* (3) pode ser transformada em um novo certificado pela adição da assinatura de I1.

AIntersect: é a interseção de duas ou mais autorizações presentes nos certificados de autorização.

VIntersect: é a interseção de duas ou mais datas de validade de certificados de autorização. É obtida da seguinte forma:

Para X = (Xi, Xs) e Y = (Yi, Ys)

VIntersect (X, Y) = (Vmin, Vmax)

Onde:

Vmin = max (Xi, Yi)

Vmax = min (Xs, Ys)

Além disso,

VIntersect (X, Y, Z,...) = VIntersect (X, VIntersect (Y, Z,...))

Caso *Vmin* > *Vmax* a interseção falha.

4.5. Funcionamento da autorização por certificados SPKI

No exemplo a seguir, será ilustrado como Alice monta uma cadeia de certificados para obter acesso de leitura de arquivos em um servidor.

Para provar que possui a autorização e acessar o recurso, Alice precisa combinar os certificados que possui, de forma a provar o que está sendo exigido pelo servidor. Ela repete a sua requisição, incluindo a *certificate chain* (cadeia de certificados) para demonstrar que a sua chave pública foi autorizada e assina a requisição com a sua chave secreta correspondente a esta chave pública.

Detalhadamente, o funcionamento de um processo de autorização utilizando certificados SPKI ocorre conforme descrito a seguir, considerando:

Chave autorizada: K_S

Certificados do cliente:

{K_S, K_G, verdadeiro, leitura e escrita de arquivos, 10/05/2003 – 20/05/2003} (1)

{K_G, usuario, falso, leitura de arquivos, 10/05/2003 – 15/05/2003 (2)

{K_G, usuario, Alice, 01/05/2003 – 01/06/2003} (3)

{K_G, Alice, K_A, 01/05/2003 – 01/06/2003} (4)

onde:
K_S = chave pública do servidor
K_G = chave pública do gerente
K_A = chave pública da Alice
(1) e (2) são certificados de autorização
(3) e (4) são certificados de nomes

- O cliente (equipamento de Alice) envia uma requisição de acesso ao servidor.

- O servidor responde com uma negativa de acesso, juntamente com a lista de chaves autorizadas $\{K_S\}$ e solicita que o cliente comprove que o seu usuário tem direito de acesso através de certificados.
- A partir da lista de chaves autorizadas enviadas pelo servidor, o cliente faz uma busca em seus certificados SPKI e tenta formar uma cadeia de certificados que se origine com a chave autorizada (K_S) e termine com a chave de Alice (K_A). Neste caso, esta cadeia origina-se no certificado (1), onde a chave do servidor (K_S) concede autorização à chave do gerente (K_G). O próximo certificado na cadeia é o (2) onde a chave do gerente (K_G) concede autorização a um grupo chamado *usuário*. O terceiro certificado na seqüência é o (3), onde a chave do gerente (K_G) garante que Alice faz parte do grupo *usuário*. Finalizando a cadeia aparece o certificado (4), onde a chave do gerente (K_G) garante que K_A é a chave pública de Alice. Portanto, após identificar os certificados e compor a cadeia, esta teria a seguinte seqüência: (1) (2) (3) (4). O cliente então, envia uma mensagem ao servidor contendo a requisição de acesso assinada com a chave privada de Alice e a cadeia de certificados.
- Após receber a mensagem do cliente, o servidor realiza as seguintes verificações: analisa se a seqüência da cadeia de certificados, representada por (1) (2) (3) (4), está correta; se a chave pública indicada como autorizada pela cadeia de certificados corresponde à chave privada que assinou a requisição de acesso; se os direitos concedidos a Alice contidos no certificado estão coerentes com a sua solicitação e se o certificado está dentro do período de validade.

Caso todas as verificações retornem um resultado correto, Alice estará autorizada a acessar o recurso solicitado.

4.6. Conclusão

Neste capítulo foi apresentada a infra-estrutura de chave pública SPKI, abordando os seus certificados de nomes e autorização, suas características como a possibilidade de delegação, e o funcionamento de um processo de autorização utilizando certificados SPKI.

Por suas características, a SPKI demonstrou ser um tipo de infra-estrutura de chave pública com grande potencial de utilização, principalmente em ambientes onde sejam necessárias comunicações de curta duração que conseqüentemente necessitem de processos rápidos para a autorização.

Capítulo 5

Uso de Cadeias de Autorização SPKI na Segurança de Redes sem Fio IEEE 802.11

Dentro do ambiente de redes *wireless* padrão IEEE 802.11, que foi o foco principal deste estudo, a primeira preocupação foi tentar fornecer uma forma de segurança que possibilitasse aos dispositivos participantes da rede comunicarem-se com um nível de privacidade equivalente ao existente em redes fixas (*wired*), onde isto ocorre pela própria limitação física de acesso (um dispositivo somente pode comunicar-se com outro pertencente à rede se tiver acesso físico a um cabo que lhe permita trocar dados com os demais participantes dessa rede). Como dentro do ambiente *wireless* IEEE 802.11 esta limitação física não é possível, pois os sinais de rádio podem atravessar paredes e ser captados por um dispositivo em um prédio vizinho, por exemplo, a solução inicialmente encontrada foi a criação do WEP (*Wired Equivalent Privacy*). Entretanto, a segurança proporcionada pelo WEP é bastante rudimentar (baseia-se na criptografia por chave simétrica compartilhada entre os dispositivos comunicantes) e pode ser facilmente quebrada, conforme descrito em [9].

Como alternativas para melhorar a segurança foi proposta a utilização de mecanismos complementares de segurança nas camadas superiores, a fim de agregar maior confiabilidade às comunicações *wireless*. Uma das propostas mais aceitas foi a utilização do EAP (*Extensible Authentication Protocol*), que por tratar-se de um protocolo geral para autenticação possibilita a utilização de diversos mecanismos para este fim, como senhas, *smart cards* e chaves públicas.

Dentre as aplicações do EAP desenvolvidas até o momento, uma das que proporciona maior segurança e aceitação, podendo ser encontrada incorporada a softwares no mercado é a utilização do EAP juntamente com o TLS (*Transport Level Security*), chamado de EAP-TLS.

A utilização do TLS adiciona ao EAP segurança na camada de transporte, possibilitando autenticação mútua e negociação de parâmetros para o estabelecimento de uma comunicação segura entre dispositivos através da negociação de cifradores e troca de chaves. A forma utilizada para realizar a autenticação mútua entre cliente e servidor de autenticação dentro de um ambiente IEEE 802.11 com EAP-TLS é através de certificados padrão X.509.

Apesar da boa segurança proporcionada pela utilização de certificados X.509 para a autenticação, tanto do cliente para o servidor, quanto no sentido inverso, e do controle de acesso realizado por ACLs tradicionais, por exemplo, foram identificadas algumas limitações da sua utilização em ambientes dinâmicos, como é o caso de uma rede *wireless*. Dentre estas limitações, as principais foram as seguintes:

- Falta de flexibilidade pela dependência de CAs, onde existe uma centralização e hierarquização muito forte, pois os certificados somente podem ser emitidos por autoridades certificadoras que pertençam a uma lista de CAs confiáveis, que por sua vez podem solicitar a emissão de certificados a CAs hierarquicamente superiores. Portanto, sempre que for necessária a emissão de um novo certificado X.509, ela deve ser solicitada a uma CA pré-definida, sendo que este processo pode ser demorado, tanto pela distância física (a solicitação pode ser encaminhada a outras CAs) quanto pela impossibilidade de um pronto atendimento por parte de uma CA específica (indisponibilidade).

- Impossibilidade de delegação a fim de descentralizar a concessão de autorizações. Este é um ponto importante porque esta descentralização possibilita uma maior flexibilidade e agilidade ao tornar possível que se possa recorrer a elementos alternativos; como conseqüência pode haver uma economia de tempo, que é um fator importante em redes dinâmicas e de curta duração.

- Complexidade de administração de controle de acesso, que de maneira geral é feita com base no nome ou identificador de rede do usuário, o que em redes grandes e complexas acaba acarretando listas gigantescas de usuários com as respectivas autorizações de acesso, tornando a suas manutenções trabalhosas e mais suscetíveis a erros.

Tentando suprir estas limitações acima identificadas, surgiu a idéia de se propor uma melhoria na arquitetura existente (que se utiliza de IEEE 802.11 com EAP-TLS) pela introdução de certificados SPKI (*Simple Public Key Infrastructure*) em substituição aos tradicionais certificados X.509 (tanto para autenticação quanto autorização de usuários), o que possibilita a descentralização de autoridade através de delegações, bem como maior facilidade no controle de acesso (autorizações) realizado através de certificados e chaves neles contidas. Esse processo simplifica as listas de autorizações armazenadas no servidor de autenticação/autorização, pois os próprios certificados contêm as informações de autorização concedidas para aquela chave (de usuário).

5.1. Resumo da Proposta

A segurança em redes IEEE 802.11, por tratar-se de um aspecto ainda em desenvolvimento e que provavelmente necessitará de constantes atualizações, em função da rápida evolução de equipamentos e programas destinados a burlá-la, será o foco deste trabalho.

Dentro do aspecto segurança será abordada a melhoria nas formas de autenticação e autorização de usuários, proporcionando mais agilidade e flexibilidade a fim de adequar-se melhor às características de uma rede *wireless* temporária.

Seguindo este raciocínio, a estrutura de segurança proposta estará baseada na utilização dos protocolos EAP e TLS (a fim de complementar a segurança abordada nas camadas inferiores) juntamente com a infra-estrutura do SPKI que neste caso possibilitará uma melhoria na segurança, adicionado autorização de usuário por certificados ao processo de autenticação fornecido pelo EAP.

O funcionamento desta estrutura de segurança proposta será em grande parte semelhante ao funcionamento do próprio EAP-TLS [2], diferenciado-se por estar sendo utilizada em uma rede *wireless* ao invés de uma rede *wired* e pela adoção da infra-estrutura SPKI para autenticação e autorização de usuário (em substituição a uma infra-estrutura PKI tradicional). Neste caso, o tipo de autorização contido nos certificados SPKI indicará a sub-rede que o usuário poderá acessar.

O trabalho desenvolvido inicia sua abordagem a partir da associação IEEE 802.11 (descrita em [28]) finalizada e avança até a conclusão da comunicação EAP-TLS que indicará

se a autenticação mútua entre usuário e servidor de autenticação foi concluída (e o usuário está autorizado) ou interrompida devido a alguma falha.

Nesta dissertação, o trabalho apresentado baseia-se na comunicação entre três tipos de elementos, seguindo o padrão IEEE 802.11:

- *Authenticanting Peer* (*peer*): estação *wireless* utilizada pelo usuário que deseja comunicar-se com o AS para autenticar-se a ele e obter autorização de acesso.
- *Authentication Server* (AS): elemento pertencente à rede fixa (*wired*), responsável pela autenticação e autorização do usuário do *peer*.
- *Authenticator* (AP): elemento que possui interfaces com as redes *wireless* (*peers*) e *wired* (AS), e atua como intermediador entre *peers* e AS, possibilitando a comunicação entre eles.

A interação entre os três tipos de elementos ocorre conforme representado na figura a seguir.

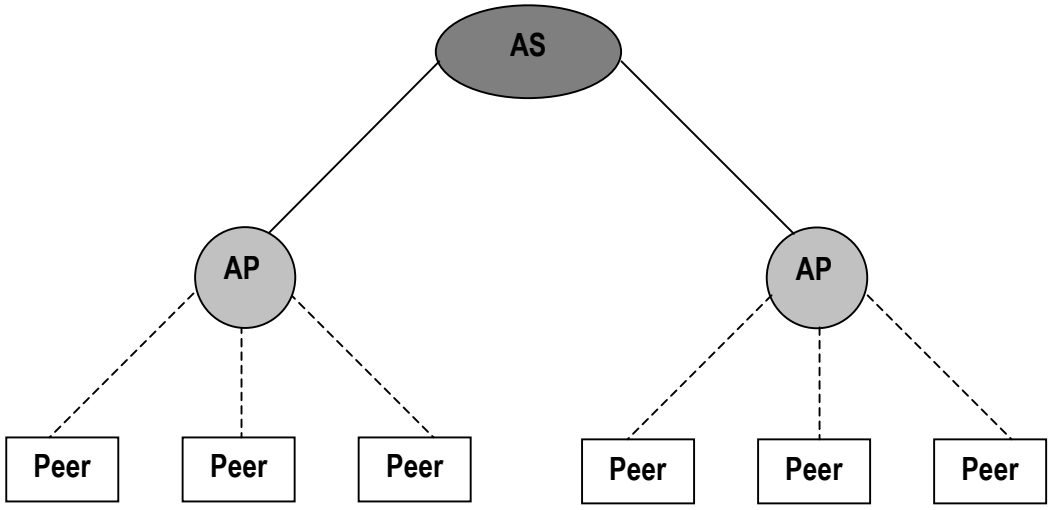

Figura 5.1 - Interação entre os elementos da proposta

5.2. Benefícios

A definição da estrutura deste trabalho levou em consideração as melhorias que poderiam ser propostas em relação aos mecanismos de segurança existentes. Dentre os benefícios identificados, os principais foram:

- Descentralização, maior flexibilidade e agilidade na emissão de certificados: isto ocorre pela independência de CAs com a conseqüente possibilidade de emissão de certificados de autorização por diversos elementos distintos pertencentes à rede (desde que possuam certificados delegáveis), permitindo que as solicitações de emissões de certificados possam ser feitas a outros elementos, caso um deles não esteja disponível, por exemplo, ou que seja escolhido um possível emissor de certificado que esteja fisicamente mais próximo para que a comunicação seja mais rápida.

- Concessão de autorizações por tempos definidos: a utilização dos certificados SPKI permite que sejam emitidos certificados de autorização por tempos definidos. Por exemplo, pode-se emitir um certificado não delegável que conceda a um visitante autorização de acesso a uma sub-rede local por apenas um dia ou algumas horas.

- Possibilidade de delegação de autorizações: esta característica permite que as autorizações concedidas a um elemento possam ser repassadas a outros (desde que possuam o poder de delegação), podendo esta ser concedida total ou parcialmente. A delegação de autorizações contribui para a descentralização e maior agilidade na emissão de certificados.

- Autorizações concedidas através de certificados/chaves públicas: tanto as autorizações fornecidas diretamente a chaves, armazenadas no servidor de autenticação, quanto as contidas nos certificados de autorização são concedidas baseando-se em chaves, o que facilita a sua administração, pois uma chave pode ser compartilhada por um grupo de usuários. No primeiro caso estão incluídas as chaves de controle do recurso, que podem ter acesso diretamente a ele sem a necessidade de certificados, e são as chaves de origem em uma cadeia de certificados. No segundo caso o tipo de autorização que determinada chave possui está contida em um certificado, derivado da autorização concedida pela chave de origem. Outro ponto importante é o fato de as autorizações estarem contidas nos certificados de autorização, o que ocasiona uma maior facilidade de administração e economia de espaço de armazenamento, pois o servidor de autenticação necessita

armazenar somente as chaves controladoras de recursos, sendo as demais chaves autorizadas através de delegações a partir destas.

- Comunicação segura através de TLS: possibilita que sejam utilizadas as características do TLS para a definição de parâmetros de segurança entre a *mobile STA* (estação móvel) do usuário e o servidor de autenticação, além de permitir realizar a autenticação mútua entre eles através de certificados X.509 (do servidor para o usuário) e certificados SPKI (do usuário para o servidor).

Portanto, os benefícios acima descritos demonstram que o trabalho proposto apresenta novas características, tornando-o diferente de infra-estruturas de segurança para redes móveis já apresentadas.

5.3. Deficiências

Além dos benefícios, também foram identificados alguns possíveis pontos negativos dentro da proposta apresentada. Dentre eles, os principais foram:

- Possível tempo adicional gasto na composição da cadeia de certificados SPKI: dependendo da quantidade de certificados que o usuário possua, esta composição de cadeia de certificados pode demorar um tempo considerável para ser montada. Entretanto, este não é um problema exclusivo do trabalho proposto e está presente nas aplicações da infra-estrutura de certificados SPKI. Para resolvê-lo, são apresentadas possíveis soluções em [14] e [15].

- Necessidade de alterações nas mobile STAs de usuários e no servidor de autenticação para suportar certificados SPKI: como o EAP-TLS foi criado para trabalhar com certificados X.509, a substituição destes por certificados SPKI (por parte do usuário) acabam acarretando algumas alterações tanto do lado da *mobile STA* do usuário (utilização de algoritmos específicos para a criação da cadeia de certificados) quanto do lado do servidor de autenticação (criação de um lista de

chaves autorizadas e análise da cadeia de certificados enviada pela *mobile STA* do usuário).

- Processamento adicional exigido por utilizar criptografia de chave pública: dependendo da capacidade de processamento e memória dos equipamentos móveis utilizados pelo usuário, o tempo de processamento gasto para cifrar/decifrar utilizando chaves públicas pode ser um fator negativo. Este aspecto tende a ser minimizado com a evolução tecnológica dos dispositivos móveis.

Considerando ambientes onde haja equipamentos móveis com boa capacidade de processamento e memória, as deficiências anteriormente apresentadas tendem a ser minimizadas. Entretanto, em ambientes onde haja dispositivos móveis com capacidade de processamento e memória restritas, as deficiências apresentadas podem ocasionar sérios problemas de desempenho, pois o tempo de espera para a elaboração da cadeia de certificados, por exemplo, pode ser longo.

5.4. Trabalhos Correlatos

Em [31], foi proposta uma integração entre certificados SPKI e TLS, sugerindo alterações nas mensagens TLS (*Certificate Request* e *Client Certificate*) e a extensão do campo *ClientCertificateType* para possibilitar a autorização do cliente via certificados SPKI, com a autenticação do servidor continuando sendo realizada através de certificados X.509. O foco deste *draft* é a adaptação de mensagens TLS para suportar do lado cliente certificados SPKI em substituição a certificados X.509.

O que foi definido em [31] difere do que está sendo proposto neste trabalho pelo fato de ser apenas conceitual (não especifica detalhes para sua implementação em nenhum tipo de ambiente), não abordar roteiros detalhados de funcionamento que retratem a sua aplicabilidade, além de tratar somente de TLS e SPKI sem abordar a utilização do EAP.

Em [10], foram apresentados os protocolos *device-to-proxy* para dispositivos leves *wireless* e *proxy-to-proxy* baseado em SPKI/SDSI, criados com o objetivo de permitir acesso de rede seguro a dispositivos móveis, incluindo os que possuem pouca capacidade de

processamento (dispositivos leves). Para permitir que dispositivos leves possam também utilizar-se da segurança proporcionada por uma infra-estrutura de chave pública, foi criado um software *proxy* em cada dispositivo. Os dispositivos comunicam-se com seus *proxies* através do protocolo *device-to-proxy* e a comunicação entre *proxies* é feita utilizando o protocolo seguro *proxy-to-proxy* baseado em SPKI/SDSI.

Existem várias diferenças entre o que foi definido em [10] e o que está sendo proposto neste trabalho como a ausência de autenticação do servidor e de confidencialidade (criptografia) dos dados trocados entre cliente e servidor (em [10]), que podem ser obtidos pela utilização de TLS ou EAP-TLS.

Em [14] foi apresentada a parte servidor de um projeto denominado *Geronimo* que explora a viabilidade de se utilizar SPKI/SDSI para fornecer controle de acesso pela Web, além de descrever um algoritmo para *certificate chain discovery* em SPKI/SDSI. Também é sugerida uma possível utilização de SSL/TLS como forma de incrementar a segurança proposta.

Como diferenças entre o que foi definido em [14] e o que está sendo proposto aqui, vale salientar o fato do primeiro concentrar-se em um ambiente de rede fixa, tratar o TLS como uma possível complementação à segurança sem definir detalhes para a sua implementação e abordar definição de um algoritmo para *certificate chain discovery*, sendo que a descoberta de cadeias de certificados não faz parte do escopo do trabalho aqui proposto.

Em [15] foi apresentado um algoritmo para resolver o problema denominado *certificate chain discovery*. Através deste algoritmo é possível a montagem da *certificate chain* a partir dos certificados SPKI possuídos pelo usuário.

O algoritmo apresentado em [15] serve como referência para a resolução de um problema comum encontrado nas diversas aplicações da infra-estrutura SPKI, e portanto, não aborda um problema específico do trabalho proposto. Por isso, este algoritmo serve como fonte de consulta para a descoberta e montagem de cadeias de certificados SPKI, citadas neste trabalho sem maior detalhamento por não fazer parte do seu escopo.

Em [26] foi especificado um mecanismo de autenticação e geração de chaves de sessão em um ambiente de *roaming*, utilizando-se EAP e TLS. Neste caso, a autenticação do servidor e a negociação de chave de sessão são realizadas usando EAP TLS e a autenticação do usuário é feita utilizando qualquer mecanismo EAP definido em [7]. Também discute de maneira geral uma possível utilização do mecanismo proposto em redes IEEE 802.11.

O mecanismo de autenticação definido em [26] baseia-se na utilização convencional do EAP-TLS, utilizando-se de certificados X.509 para a autenticação mútua. A possível utilização deste mecanismo em redes IEEE 802.11 é apresentada de maneira conceitual e sem detalhamentos. Portanto, apresenta um foco diferente do proposto neste trabalho.

5.5. Descrição Detalhada

O trabalho desenvolvido envolve a proposição de uma melhoria na arquitetura de segurança das redes IEEE 802.11 a partir da camada 3 do modelo OSI, visando complementar a segurança proporcionada pela camada 2 (abordada na especificação do padrão IEEE 802.11) utilizando para isso EAP e TLS, bem como proporcionar uma maior flexibilidade nas autenticações e autorizações de usuários através da utilização dos certificados de autorização SPKI, podendo estes ser delegáveis ou não, dependendo das necessidades. As autorizações, acrescentadas à autenticação EAP pela utilização de certificados SPKI, nesta proposta especificam as sub-redes as quais o usuário identificado como beneficiário no certificado de autorização SPKI pode acessar.

A proposta baseia-se na utilização de padrões/tecnologias existentes, definidos através de especificações e RFCs. Portanto, este trabalho de pesquisa não se concentra na criação de novos padrões e sim em como estes padrões/tecnologias podem ser integrados de forma a produzir um ambiente seguro para a comunicação entre os elementos (estações móveis, *access points*, servidor de autenticação) pertencentes a uma rede *wireless* padrão IEEE 802.11.

Para ilustrar a proposta, foram utilizados três roteiros, os quais serão descritos a seguir. Em cada um dos roteiros foi considerada a presença de três elementos comunicantes:

- *Authenticating Peer* (mobile STA): trata-se de uma estação móvel *wireless* com interface padrão IEEE 802.11, utilizada pelo usuário para comunicar-se com o *Authenticator* (AP) e através dele ter acesso ao *Authentication Server* para autenticação e autorização. Também pode ser referenciado apenas por *peer*. Posteriormente, após a conclusão dos processos de autenticação e autorização, o *Authenticating Peer* pode comunicar-se diretamente com o *Authenticator* (AP) e ter acesso à rede. A autenticação e comprovação de autorização do *Authenticating Peer*

para o *Authentication Server* é feita utilizando-se requisições assinadas e cadeias de certificados de autorização SPKI. Para [28] a definição de *station* (STA) é a seguinte: "qualquer dispositivo que contenha um MAC (*Medium Access Control*) conforme a especificação IEEE 802.11 e interface de camada física ao meio *wireless* (WM)". Também para [28], *mobile station* significa "um tipo de estação que usa comunicações de rede enquanto em movimento".

- *Authenticator* (AP): trata-se de uma estação com uma interface IEEE 802.11 para associar-se e comunicar-se com o *Authenticating Peer* e uma interface com a rede fixa para comunicar-se com o *Authentication Server*. O *Authenticator* é um AP (*Access Point*) que funciona como um elemento intermediário que possibilita o acesso do *Authenticating Peer* ao servidor de autenticação e à rede. Ele apenas encaminha as informações trocadas entre o *Authenticating Peer* e o *Authentication Server* (durante a autenticação e a autorização) e posteriormente entre o *Authenticating Peer* e os recursos da rede. Para [28] *access point* (AP) significa "qualquer entidade que tenha funcionalidade de estação e forneça acesso aos serviços distribuídos, através do meio *wireless* (WM) para estações associadas".

- *Authentication Server* (AS): trata-se de uma estação (servidor) conectada à rede fixa responsável pela autenticação e autorização do *Authenticating Peer*, negociação de parâmetros para o estabelecimento de conexão segura com o *Authenticating Peer*, bem como se autenticar para ele utilizando certificados padrão X.509.

O escopo deste trabalho de pesquisa corresponde ao período compreendido desde a associação estabelecida entre as estações IEEE 802.11 (camada 2 do modelo OSI) até as confirmações de que a as autenticações de cliente e servidor foram concluídas com sucesso, que o cliente está autorizado a ter acesso à rede e que os parâmetros de segurança foram definidos com sucesso para que a partir de então o cliente possa estabelecer uma conexão segura (TLS) com os recursos da rede através do *Access Point*.

Para retratar o trabalho de pesquisa desenvolvido, foram escolhidos três roteiros diferentes, que resumidamente apresentam as seguintes situações:

- Roteiro 1 (Mobile STA do usuário possui chave autorizada): neste caso, o usuário não possui certificados SPKI que comprovem a sua autorização e sim uma das chaves autorizadas pelo próprio servidor de autenticação, através da qual ele comprovará a autorização.

- Roteiro 2 (Mobile STA do usuário é capaz de gerar a SPKI *certificate chain*): neste caso, o usuário possui os certificados SPKI que comprovam a sua autorização e então a sua mobile STA monta a cadeia de certificados SPKI e os envia ao servidor de autenticação.

- Roteiro 3 (Mobile STA do usuário não possui chave autorizada nem é capaz de gerar SPKI *chain*): neste caso, o usuário inicialmente não possui certificados SPKI que comprovem a sua autorização nem chave autorizada pelo próprio servidor de autenticação. Adicionalmente, são propostos dois roteiros para que o usuário possa obter o certificado de autorização desejado.

Vale salientar que estes roteiros não são os únicos possíveis. Existem outros casos que podem ser abordados a fim de complementar as explicações apresentadas, bem como para retratar outras situações. Estes três roteiros foram escolhidos por possibilitarem retratar situações importantes dentro do ambiente considerado, além de permitir uma descrição completa do funcionamento da infra-estrutura proposta, incluindo as informações trocadas entre os elementos envolvidos.

A seguir, serão descritos detalhadamente os três roteiros anteriormente citados.

Roteiro 1: *Mobile STA* do usuário possui chave autorizada

Neste primeiro roteiro, foi considerado um caso em que o *Authenticating Peer* identifica-se e solicita ao servidor de autenticação acesso a uma sub-rede. O servidor então informa à *mobile STA* do usuário que ele deve comprovar que possui uma chave autorizada para o acesso através de uma SPKI *certificate chain* (cadeia de certificados), originada a partir de uma das chaves autorizadas pelo servidor e contida na lista por ele enviada. A *mobile STA*

possuindo uma das chaves autorizadas, envia uma cadeia de certificados SPKI vazia ao servidor de autenticação, sendo que a comprovação de autorização do usuário (chave pública) é feita pela identificação de que ele possui a chave privada e a respectiva chave pública autorizada.

Para facilitar o seu entendimento, o roteiro foi dividido em quatro passos:

- Passo 1: Identificação do usuário e início do TLS.
- Passo 2: Solicitação de acesso do usuário e envio de lista de chaves e certificado pelo servidor.
- Passo 3: Autenticação do servidor e autenticação/autorização do usuário.
- Passo 4: Confirmação/finalização da autenticação e autorização.

Após a associação IEEE 802.11 entre o Authenticating Peer (*peer*) e o Authenticator (AP), tem início o EAP-TLS. Para maiores detalhes sobre a fase de associação, consultar [28].

Passo 1: Identificação do usuário e início do TLS

Conforme descrito em [2], o início da conversação EAP-TLS tipicamente inicia-se com a negociação do EAP.

1. O AP envia um pacote `{EAP-Request/Identity}` solicitando à *mobile STA* a identificação do usuário.
2. A *mobile STA* responde com um pacote `{EAP-Response/Identity}`, informando ao AP a identidade do usuário através da sua chave pública.
3. O AP encaminha a resposta recebida do *Peer* para o servidor de autenticação.
4. Após receber a identidade do usuário, o AS envia um pacote `{EAP-Request/EAP-TLS (TLS Start)}`, indicando o início do TLS.
5. O AP encaminha para o *Peer* o pacote recebido do AS em (4).

Passo 2: Solicitação de acesso do usuário e envio de lista de chaves e certificado pelo servidor

Após o *Peer* receber o pacote indicando o início do TLS em (5), a comunicação continua da seguinte forma:

6. *A mobile STA* responde com pacote `{EAP-Response/EAP-TLS (TLS client-hello, Requisição)}`. A mensagem `TLS client_hello` contém informações para a definição do TLS (versão do TLS da *mobile STA*, *sessionID*, número aleatório gerado pela *mobile STA* e cifradores suportados por ela). A mensagem `Requisição` trata-se de uma solicitação de acesso SPKI, através da qual a *mobile STA* do usuário solicita ao servidor de autenticação permissão de acesso a uma sub-rede.
7. As informações contidas em (6) são encaminhadas pelo AP até o AS.

A. O AS após receber a requisição de acesso, a analisa e após consultar e identificar as chaves públicas autorizadas para o tipo de acesso solicitado pelo *Peer*, gera uma **lista de chaves autorizadas**.

8. O AS envia um pacote `{EAP-Request/EAP-TLS (TLS server_hello, TLS certificate, TLS server_key_exchange, TLS certificate_request, TLS server_hello_done, Requisição)}`. A mensagem `TLS server_hello` contém: número da maior versão do TLS suportada pelo servidor, número aleatório gerado pelo servidor, *sessionID* e *cipher suite* selecionada pelo servidor. A mensagem `TLS certificate` é a cadeia de certificados X.509 para a autenticação do servidor. A mensagem `TLS server_key_exchange` é opcional e utilizada para fornecer informações criptográficas à *mobile STA* para a comunicação do *premaster secret* (segredo primário) somente quando o *server certificate* não fornece dados suficientes ao cliente para a troca do segredo primário. A mensagem `TLS certificate_request` é utilizada para solicitar a autenticação do usuário através de certificados. Neste caso, foi realizada uma alteração na mensagem `TLS certificate_request`, substituindo a lista de CAs aceitáveis nele contida por uma lista de chaves públicas autorizadas. A mensagem `Requisição` trata-se de uma cópia da própria requisição de acesso enviada pelo *Peer* em (6).

9. As informações contidas em (8) são encaminhas pelo AP para o *Peer*.

Passo 3: Autenticação do servidor e autenticação/autorização do usuário

Após o *Peer* receber o pacote contendo informações enviadas pelo servidor de autenticação em (9), a comunicação continua da seguinte forma:

B. O *Peer* primeiramente analisa o `TLS certificate` enviado pelo AS para autenticar o servidor. Depois, ele analisa a lista de chaves autorizadas recebidas, identifica que possui uma das chaves autorizadas, gera uma `SPKI certificate chain vazia` e um `timestamp`. Então, a mobile STA cria uma `seqüência SPKI` contendo a `requisição` e o `timestamp`, assinando-a com a chave privada do usuário e inclui uma cópia da chave pública do usuário na assinatura SPKI.

10. O *Peer* envia um pacote`{EAP-Response/EAP-TLS (certificate, TLS client_key_exchange, TLS certificate_verify, TLS change_cipher_spec, TLS finished)}`. O `certificate` neste caso, não é um certificado padrão X.509, mas sim composto pela `SPKI certificate chain (vazia)` e pela `seqüência SPKI` (criados no item (B)), na forma `((requisição, timestamp) Ku, SPKI certificate chain)`, onde Ku é a chave do usuário. A mensagem `TLS client_key_exchange` é utilizada para a definição do segredo primário entre o cliente (*mobile STA*) e o servidor de autenticação, através da transmissão direta do segredo criptografado ou pela transmissão de parâmetros. A mensagem `TLS certificate_verify` é uma resposta de autenticação assinada pelo cliente, utilizado para a verificação do certificado do usuário. A mensagem `TLS change_cipher_spec` é utilizada pelo *peer* (*mobile STA*) para notificar o servidor sobre a mudança de especificação de cifragem para as comunicações subseqüentes. A mensagem `TLS finished` contém uma resposta de autenticação do *peer* para o servidor. É enviada para verificar que a *key exchange* e o processo de autenticação foram realizados com sucesso.

11. As informações contidas em (10) são encaminhas pelo AP para o AS.

C. O AS primeiramente verifica o `certificate` recebido em (11): analisa a validade do `timestamp`, verifica se a `requisição` recebida é a mesma enviada

inicialmente pelo cliente, extrai a chave pública da assinatura, verifica a assinatura na `sequência requisição-timestamp` utilizando a chave pública obtida. Como a `SPKI certificate chain` recebida está vazia, o AS neste caso compara a chave pública do usuário com a lista de chaves públicas autorizadas do próprio servidor para poder garantir a autenticação/autorização do usuário. Em seguida, o AS analisa os parâmetros TLS enviados pelo *Peer* para confirmar se estão coerentes e então validá-los.

12. Após confirmada a autenticação (através da assinatura da `seqüência requisição-timestamp`), a autorização da chave do usuário (através da localização da chave pública do usuário na lista de chaves autorizadas) e validados os parâmetros TLS, o AS envia um pacote `{EAP-Request/EAP-TLS (TLS change_cipher_spec, TLS finished)}`. A mensagem `TLS change_cipher_spec` contém a confirmação da troca de especificação de cifragem. A mensagem `TLS finished` contém uma resposta de autenticação do servidor para o *peer*. É enviada para verificar que a *key exchange* e o processo de autenticação foram realizados com sucesso.
13. As informações contidas em (12) são encaminhas pelo AP para o *Peer*.

Passo 4: Confirmação/ finalização da autenticação e autorização

Após o *Peer* receber o pacote contendo informações enviadas pelo servidor de autenticação em (13), a comunicação continua da seguinte forma:

D. O *Peer* analisa as informações TLS recebidas do AS e se estiverem coerentes finaliza com sucesso a autenticação do servidor.
14. Para confirmar a autenticação do AS e o recebimento de (13), o *Peer* envia um pacote `{EAP-Response/EAP-TLS (vazio)}`.
15. As informações contidas em (14) são encaminhas pelo AP para o AS.
16. O AS envia um pacote `{EAP-Success}`, indicando que o processo de autenticação/autorização foi concluído com sucesso. Neste caso, o AS envia para o AP (de forma segura) a chave de sessão negociada com o *Peer* para que a partir de então, cliente e AP possam comunicar-se diretamente.

17. O pacote {EAP-Success} é encaminhado pelo AP para o *Peer* e deste ponto em diante (dentro do período de validade da sessão) os dois poderão comunicar-se sem a presença do AS.

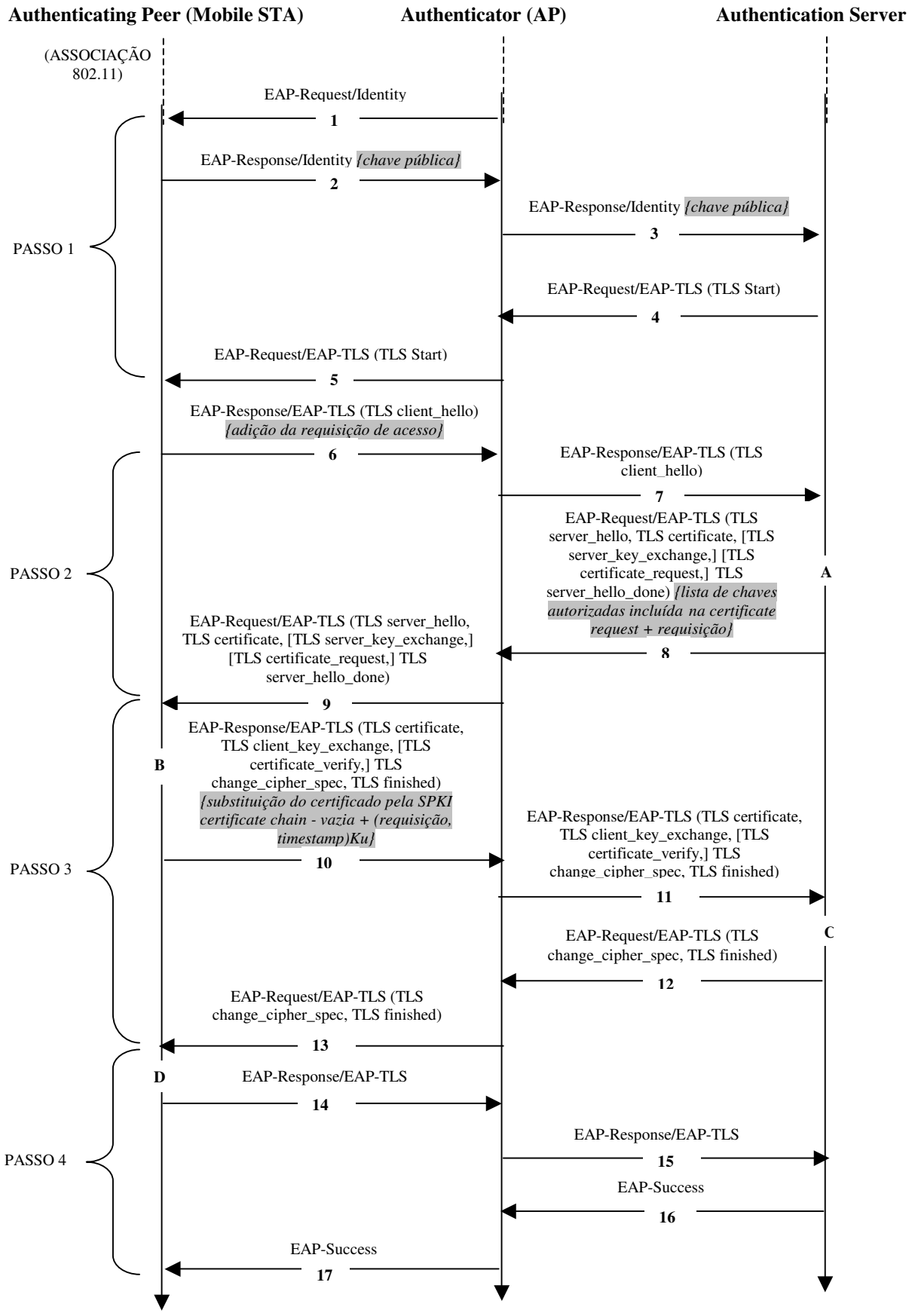

Figura 5.2 - Funcionamento do Roteiro 1

Roteiro 2: *Mobile STA* do usuário é capaz de gerar a SPKI *certificate chain*

Neste segundo roteiro, foi considerado um caso em que o *Authenticating Peer* identifica-se e solicita ao servidor de autenticação acesso a uma sub-rede. O servidor então informa à *mobile STA* do usuário que ele deve comprovar que possui uma chave autorizada para o acesso através de uma SPKI *certificate chain* (cadeia de certificados), originada a partir de uma das chaves autorizadas pelo servidor e contida na lista por ele enviada. A *mobile STA* possuindo os certificados necessários, monta a cadeia de certificados SPKI e a envia ao servidor de autenticação comprovando que o usuário (chave pública) está autorizado.

Para facilitar o seu entendimento, o roteiro foi dividido em quatro passos:

- Passo 1: Identificação do usuário e início do TLS.
- Passo 2: Solicitação de acesso do usuário e envio de lista de chaves e certificado pelo servidor.
- Passo 3: Autenticação do servidor e autenticação/ autorização do usuário.
- Passo 4: Confirmação/ finalização da autenticação e autorização.

Os passos 1 e 2 são idênticos aos descritos no roteiro 1.

Passo 3: Autenticação do servidor e autenticação/autorização do usuário

Após o *Peer* receber o pacote contendo informações enviadas pelo servidor de autenticação em (9), a comunicação continua da seguinte forma:

B. O *Peer* primeiramente analisa o `TLS certificate` enviado pelo AS para autenticar o servidor. Depois, ele analisa a **lista de chaves autorizadas** recebidas e gera a `SPKI certificate chain`, utilizando um *certificate chain discovery algorithm* [14, 15], que utiliza como entradas a `lista de chaves`, a `requisição`, a chave pública do usuário, o conjunto de certificados do usuário e um `timestamp` criado pelo *Peer*. Ele cria uma `seqüência SPKI` contendo a `requisição` e o `timestamp`, assinando-a com a chave privada do usuário e inclui uma cópia da chave pública do usuário na assinatura SPKI.

10. O *Peer* envia um pacote `{EAP-Response/EAP-TLS (certificate, TLS client_key_exchange, TLS certificate_verify, TLS`

change_cipher_spec, TLS finished)}. O certificate neste caso, não é um certificado padrão X.509, mas sim composto pela SPKI certificate chain e pela seqüência SPKI (criados no item (B)), na forma ((requisição, timestamp) Ku, SPKI certificate chain), sendo Ku a chave do usuário. A mensagem TLS client_key_exchange é utilizada para a definição do segredo primário entre o *peer* (*mobile STA*) e o servidor de autenticação, através da transmissão direta do segredo criptografado ou pela transmissão de parâmetros. A mensagem TLS certificate_verify é uma resposta de autenticação assinada pelo cliente, utilizado para a verificação do certificado do usuário. A mensagem TLS change_cipher_spec é utilizada pelo *peer* para notificar o servidor sobre a mudança de especificação de cifragem para as comunicações subseqüentes. A mensagem TLS finished contém uma resposta de autenticação do *peer* (*mobile STA*) para o servidor. É enviada para verificar que a *key exchange* e o processo de autenticação foram realizados com sucesso.

11. As informações contidas em (10) são encaminhas pelo AP para o AS.

C. O AS primeiramente verifica o certificate recebido em (11): analisa a validade do timestamp, verifica se a Requisição recebida é a mesma enviada inicialmente pelo *peer*, extrai a chave pública da assinatura, verifica a assinatura na sequência requisição-timestamp utilizando a chave pública obtida, valida os certificados da SPKI certificate chain, verifica se há uma cadeia de autorização originando em uma das chaves autorizadas (enviadas pelo próprio AS) e chegando até a chave contida na assinatura através da SPKI certificate chain apresentada. Em seguida, o AS analisa os parâmetros TLS enviados pelo *Peer* para confirmar se estão coerentes e então validá-los.

12. Após confirmada a autenticação (através da assinatura da seqüência requisição-timestamp), a autorização da chave do usuário (através da SPKI certificate chain) e validados os parâmetros TLS, o AS envia um pacote {EAP-Request/EAP-TLS (TLS change_cipher_spec, TLS finished)}. A mensagem TLS change_cipher_spec contém a confirmação da troca de especificação de cifragem. A mensagem TLS finished contém uma resposta de autenticação do servidor para o *peer*. É

enviada para verificar que a *key exchange* e o processo de autenticação foram realizados com sucesso.

13. As informações contidas em (12) são encaminhas pelo AP para o *Peer*.

Passo 4: Confirmação/finalização da autenticação e autorização

Após o *Peer* receber o pacote contendo informações enviadas pelo servidor de autenticação em (13), a comunicação continua da seguinte forma:

D. O *Peer* analisa as informações TLS recebidas do AS e se estiverem coerentes finaliza com sucesso a autenticação do servidor.

14. Para confirmar a autenticação do AS e o recebimento de (13), o *Peer* envia um pacote `{EAP-Response/EAP-TLS (vazio)}`.
15. As informações contidas em (14) são encaminhas pelo AP para o AS.
16. O AS envia um pacote `{EAP-Success}`, indicando que o processo de autenticação/autorização foi concluído com sucesso. Neste caso, o AS envia para o AP (de forma segura) a chave de sessão negociada com o *Peer* para que a partir de então, cliente e AP possam comunicar-se diretamente.
17. O pacote `{EAP-Success}` é encaminhado pelo AP para o *Peer* e deste ponto em diante (dentro do período de validade da sessão) os dois poderão comunicar-se sem a presença do AS.

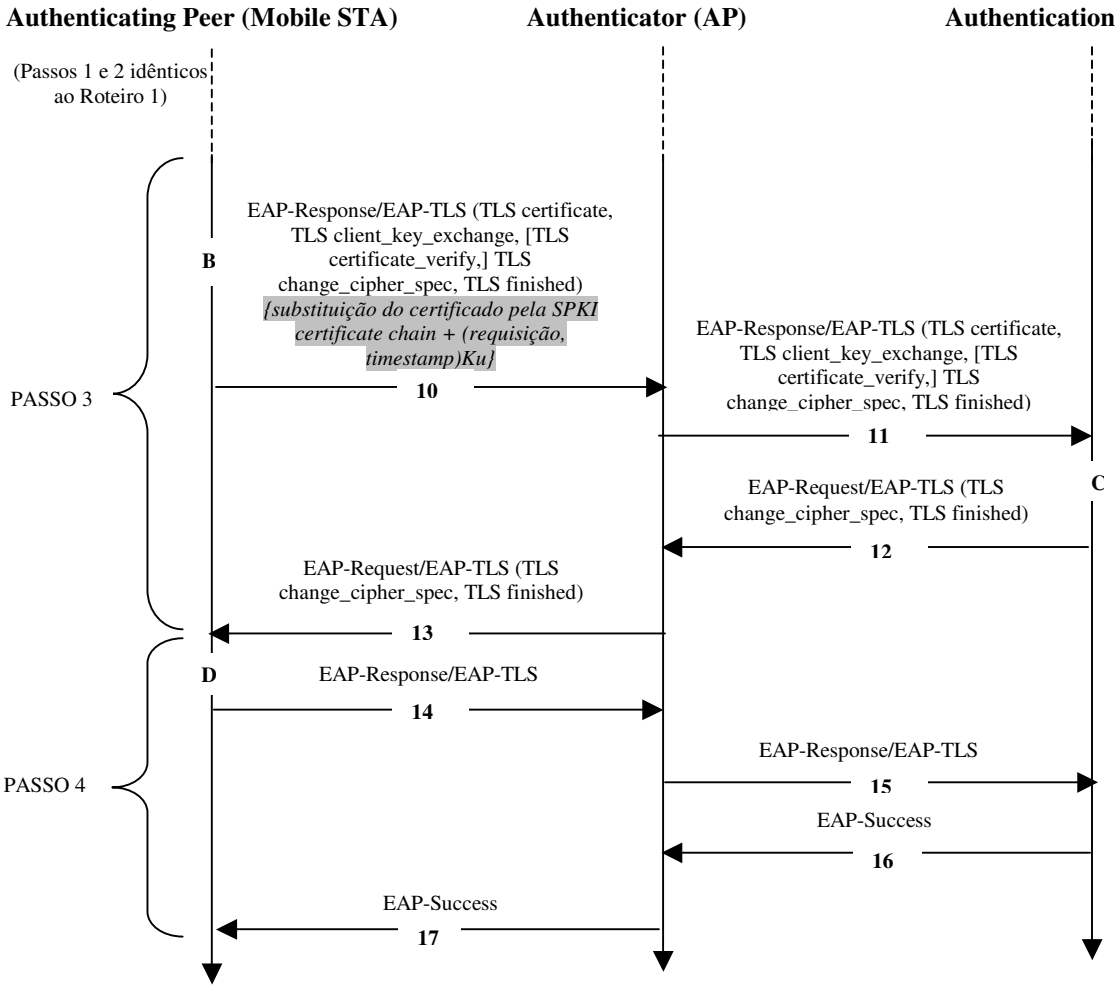

Figura 5.3 - Funcionamento do Roteiro 2

Roteiro 3: *Mobile STA* do usuário não possui chave autorizada nem é capaz de gerar SPKI *certificate chain*

Neste terceiro roteiro, foi considerado um caso em que o *Authenticating Peer* identifica-se e solicita ao servidor de autenticação acesso a uma sub-rede. O servidor então informa à *mobile STA* do usuário que ele deve comprovar que possui uma chave autorizada para o acesso através de uma cadeia de certificados SPKI, originada a partir de uma das chaves autorizadas pelo servidor e contida na lista por ele enviada. A *mobile STA* busca uma das chaves autorizadas entre os certificados que o usuário possui, e como não a encontra, envia uma cadeia de certificados SPKI vazia ao servidor de autenticação, não conseguindo assim comprovar que o usuário está autorizado, o que acaba ocasionando o envio de uma

mensagem de alerta pelo servidor de autenticação e bloqueio de acesso à rede para aquele usuário. Alternativamente, para que o usuário possa ser autorizado pelo servidor, ele deverá obter um certificado de autorização para reiniciar o processo de autenticação e autorização e então obter o acesso solicitado.

Roteiro 3.1: Falha na autorização do usuário

Neste primeiro item do roteiro 3, estão sendo abordados as mensagens e processamentos compreendidos até a confirmação de falha na autenticação/autorização do usuário, indicada pelo servidor de autenticação.

Para facilitar o seu entendimento, o roteiro foi dividido em quatro passos:

- Passo 1: Identificação do usuário e início do TLS.
- Passo 2: Solicitação de acesso do usuário e envio de lista de chaves e certificado pelo servidor.
- Passo 3: Autenticação do servidor, autenticação e falha na autorização do usuário.
- Passo 4: Confirmação de falha no processo de autenticação/autorização do usuário.

Os passos 1 e 2 são idênticos aos descritos no roteiro 1.

Passo 3: Autenticação do servidor, autenticação e falha na autorização do usuário

Após o *Peer* receber o pacote contendo informações enviadas pelo servidor de autenticação em (9), a comunicação continua da seguinte forma:

B. O *Peer* primeiramente analisa o `TLS certificate` enviado pelo AS para autenticar o servidor. Depois, ele analisa a lista de chaves autorizadas recebidas, realiza uma busca entre seus certificados para tentar encontrar uma destas chaves, mas não a encontra e gera uma `SPKI certificate chain vazia` e um `timestamp`. Então, a *mobile STA* cria uma `seqüência SPKI` contendo a `requisição` e o `timestamp`, assinando-a com a chave privada do usuário e inclui uma cópia da chave pública do usuário na assinatura SPKI.

10. O *Peer* envia um pacote `{EAP-Response/EAP-TLS (certificate, TLS client_key_exchange, TLS certificate_verify, TLS change_cipher_spec, TLS finished)}`. O `certificate` neste caso, não é um certificado padrão X.509, mas sim composto pela `SPKI certificate chain (vazia)` e pela `seqüência SPKI` (criados no item (B)), na forma `((requisição, timestamp) Ku, SPKI certificate chain)`, onde Ku é a chave do usuário. A mensagem `TLS client_key_exchange` é utilizada para a definição do segredo primário entre o peer (*mobile STA*) e o servidor de autenticação, através da transmissão direta do segredo criptografado ou pela transmissão de parâmetros. A mensagem `TLS certificate_verify` é uma resposta de autenticação assinada pelo cliente, utilizado para a verificação do certificado do usuário. A mensagem `TLS change_cipher_spec` é utilizada pelo *peer* para notificar o servidor sobre a mudança de especificação de cifragem para as comunicações subseqüentes. A mensagem `TLS finished` contém uma resposta de autenticação do *peer* para o servidor. É enviada para verificar que a *key exchange* e o processo de autenticação foram realizados com sucesso.
11. As informações contidas em (10) são encaminhas pelo AP para o AS.

C. O AS primeiramente verifica o `certificate` recebido em (11): analisa a validade do `timestamp`, verifica se a requisição recebida é a mesma enviada inicialmente pelo cliente, extrai a chave pública da assinatura, verifica a assinatura na `sequência requisição-timestamp` utilizando a chave pública obtida. Como a `SPKI certificate chain` recebida está vazia, o AS neste caso compara a chave pública do usuário com a lista de chaves públicas autorizadas do próprio servidor e como não identifica a chave do usuário como autorizada, não conclui a sua autorização. Em seguida, o AS analisa os parâmetros TLS enviados pelo *Peer* para confirmar se estão coerentes e então validá-los.

Apesar de confirmada a autenticação (através assinatura da `seqüência requisição-timestamp`), a autorização da chave do usuário não foi concluída (a chave pública do usuário não foi localizada na lista de chaves autorizadas).

12. O AS envia um pacote `{EAP-Request/EAP-TLS (TLS change_cipher_spec, TLS finished)}`. A mensagem `TLS`

`change_cipher_spec` contém a confirmação da troca de especificação de cifragem. A mensagem `TLS finished` contém uma resposta do servidor para o *Peer*. É enviada para verificar que a *key exchange* foi realizada com sucesso.

13. As informações contidas em (12) são encaminhadas pelo AP para o *Peer*.

Passo 4: Confirmação de falha no processo de autenticação/autorização do usuário

Após o *Peer* receber o pacote contendo informações enviadas pelo servidor de autenticação em (13), a comunicação continua da seguinte forma:

D. O *Peer* analisa as informações TLS recebidas do AS e se estiverem coerentes finaliza com sucesso a autenticação do servidor.

14. Para confirmar a autenticação do AS e o recebimento de (13), o *Peer* envia um pacote `{EAP-Response/EAP-TLS (vazio)}`.
15. As informações contidas em (14) são encaminhadas pelo AP para o AS.

E. O servidor de autenticação identifica o erro ocorrido no processo de autenticação/autorização do usuário (neste caso, a falta da `SPKI certificate chain` ou de chave autorizada pelo servidor que comprove a autorização do usuário) e cria uma `TLS Alert message` contendo a causa do erro.

16. Para avisar o *Peer* sobre o erro, o AS envia um pacote `{EAP-Request/EAP-TLS (TLS Alert message)}`.
17. As informações contidas em (16) são encaminhas pelo AP para o *Peer*.

F. O *Peer* analisa as informações de erro de autorização contidas em (17) e caso seja possível corrigi-lo, reinicia o processo a partir do passo 2.

18. Como não pôde ser corrigir o erro, pois o usuário não possui certificados que comprovem a sua autorização, o *Peer* envia um pacote `{EAP-Response/EAP-TLS (vazio)}` apenas para confirmar o recebimento da mensagem `TLS Alert`.
19. As informações contidas em (18) são encaminhas pelo AP para o AS.

Em (19) o recebimento de um pacote `{EAP-Response/EAP-TLS (vazio)}` pelo AS, indica que o usuário não tem como provar a sua autorização.

20. O servidor de autenticação envia um pacote `{EAP-Failure}`, indicando que a conversação foi encerrada e o processo de autenticação/autorização do usuário não foi concluído com sucesso.
21. As informações contidas em (20) são encaminhas pelo AP para o *Peer*.

Neste caso, para que o usuário do *Peer* possa ser autorizado pelo servidor de autenticação, ele deverá obter um certificado (ou cadeia de certificados) que a comprovem, utilizando-se dos roteiros descritos em 3.2 e então reiniciar a toca de informações a partir do passo 1, como descrito anteriormente.

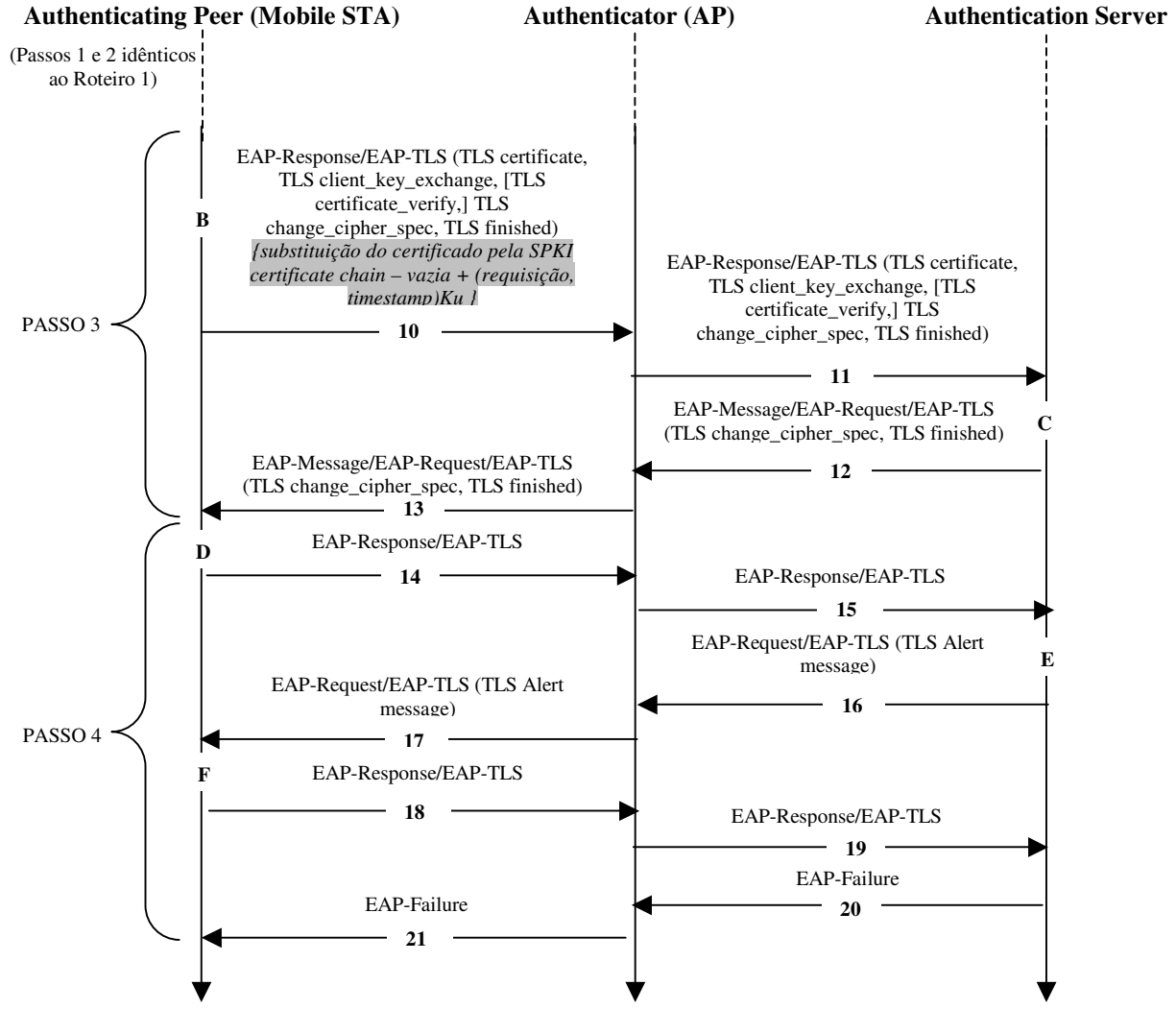

Figura 5.4 - Funcionamento do Roteiro 3.1

Roteiro 3.2: Obtenção de certificado de autorização SPKI

Neste segundo item do roteiro 3, estão sendo abordados as mensagens e processamentos de duas alternativas propostas para a obtenção de certificados de autorização SPKI, após a falha de autorização do usuário (item 3.1). Estas duas alternativas são: busca de estações que possuam chave autorizada e poder de delegação, comunicação com estação conhecida (interação pessoal prévia).

Roteiro 3.2.1: Busca de estações que possuam chave autorizada e poder de delegação

Neste roteiro, é proposta uma comunicação genérica (não foram definidos protocolos específicos) entre duas estações: a *Mobile STA Solicitante* (estação móvel do usuário que solicita a emissão de certificado de autorização) e *STA Fornecedora* (estação que poderá fornecer o certificado de autorização SPKI). Neste caso, as duas estações (e seus respectivos usuários) são desconhecidas (não haviam estabelecido conversações prévias). A *STA Solicitante* solicita informações sobre a possibilidade de geração de um certificado de autorização SPKI pela *STA Fornecedora* e caso seja possível, ela envia o certificado de autorização gerado para a *STA Solicitante*.

O detalhamento deste roteiro está descrito a seguir e esquematizado na figura abaixo:

1. A *Mobile STA Solicitante* envia uma *Requisição* de certificado de autorização SPKI para a *STA Fornecedora*. Esta requisição contém uma cópia da lista de chaves autorizadas emitida pelo servidor de autenticação no roteiro 3.1, a autorização desejada (identificação da sub-rede) e uma mensagem solicitando uma resposta que informe se a *STA Fornecedora* possui um certificado delegável emitido por uma destas chaves.

A. A *STA Fornecedora* analisa a requisição recebida e faz uma busca em seus certificados para encontrar o certificado desejado. Após encontrá-lo, analisa a sua validade e então gera uma mensagem confirmando que possui um certificado delegável que atenda as necessidades do solicitante.

2. A *STA Fornecedora* envia para a *Mobile STA Solicitante* a mensagem confirmando a possibilidade de geração do certificado.

B. Após receber a confirmação da *STA Fornecedora*, a *Mobile STA Solicitante* cria uma seqüência contendo a requisição e o período de tempo (indicando o tempo de

validade do certificado que o usuário deseja), assinando-a com a chave privada do usuário e inclui uma cópia da chave pública do usuário na assinatura.

3. A *Mobile STA Solicitante* envia a seqüência gerada em (B) para a *STA Fornecedora*.

C. A *STA Fornecedora* analisa a mensagem recebida: extrai a chave pública da assinatura, verifica a assinatura na seqüência utilizando a chave pública obtida e verifica se o período de tempo solicitado é compatível com o tempo de validade do certificado delegável que possui. Em caso afirmativo, ela emite o certificado de autorização para a chave pública do usuário, de acordo com o solicitado na mensagem (3).

4. O certificado de autorização SPKI é enviado para a *Mobile STA Solicitante*.

Após receber o certificado de autorização SPKI, a *Mobile STA Solicitante* confere-o e caso esteja correto, poderá comunicar-se novamente com o AS para comprovar a sua autorização, conforme descrito no roteiro 1.

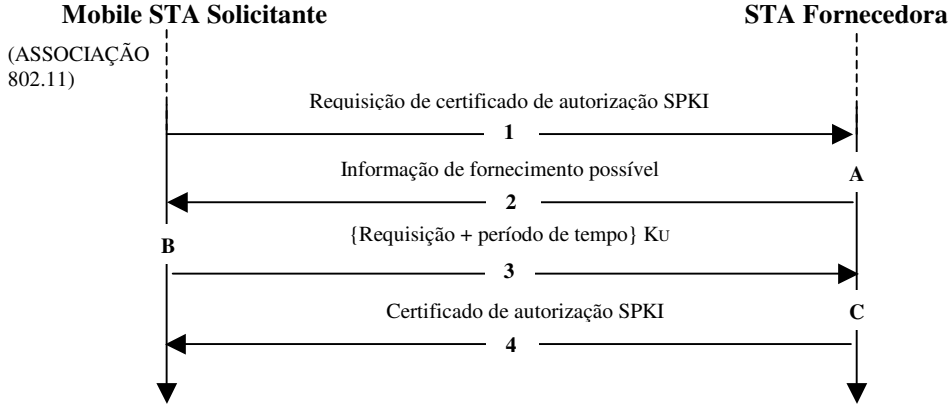

Figura 5.5 - Funcionamento do Roteiro 3.2.1

Roteiro 3.2.2: Comunicação com estação conhecida (interação pessoal prévia)

Neste roteiro, também é proposta uma comunicação genérica (não foram definidos protocolos específicos) entre duas estações: a *Mobile STA Solicitante* (estação móvel do usuário que solicita o certificado) e *STA Fornecedora* (estação que poderá fornecer o certificado de autorização SPKI). Neste caso, houve uma comunicação prévia pessoal (fora da rede) entre os usuários das duas estações, o que acaba tornando este roteiro um pouco mais

simples que o anterior. A solicitação de informações é realizada pela *STA Fornecedora* (identificada pela comunicação prévia como capaz de gerar o certificado de autorização SPKI) que solicita informações (identificação do usuário, requisição) para a emissão do certificado e após recebe-las, emite o certificado e o envia para a *STA Solicitante*.

O detalhamento deste roteiro está descrito a seguir e esquematizado na figura abaixo:

Como neste roteiro o usuário da *STA Fornecedora* já conhece informalmente o tipo de certificado que o usuário da *Mobile STA Solicitante* precisa e sabe que pode emiti-lo antes do início da comunicação entre as *STAs*, a sua *STA Fornecedora* poderá iniciar a comunicação solicitando a confirmação de informações para a emissão de certificado SPKI.

1. A *STA Fornecedora* cria uma solicitação que pede a *Mobile STA Solicitante* duas informações: uma identificação do usuário (chave pública) para quem será emitido o certificado e a requisição que contém uma cópia da lista de chaves autorizadas emitida pelo servidor de autenticação no roteiro 3.1 e a autorização desejada (identificação da sub-rede).

A. Após receber a solicitação da *STA Fornecedora*, a *Mobile STA Solicitante* cria uma seqüência contendo a requisição (lista de chaves autorizadas emitida pelo servidor de autenticação + a autorização desejada) e o período de tempo (indicando o tempo de validade do certificado que o usuário deseja), assinando-a com a chave privada do usuário e inclui uma cópia da chave pública do usuário na assinatura.

2. A *Mobile STA Solicitante* envia a seqüência gerada em (A) para a *STA Fornecedora*.

B. A *STA Fornecedora* analisa a mensagem recebida: extrai a chave pública da assinatura, verifica a assinatura na seqüência utilizando a chave pública obtida e verifica se o período de tempo solicitado é compatível com o tempo de validade do certificado delegável que possui. Em caso afirmativo, ela emite o certificado de autorização para a chave pública do usuário, de acordo com o solicitado na mensagem (2).

3. O certificado de autorização SPKI é enviado para a *Mobile STA Solicitante*.

Após receber o certificado de autorização SPKI, a *Mobile STA Solicitante* confere-o e caso esteja correto, poderá comunicar-se novamente com o AS para comprovar a sua autorização, conforme descrito no roteiro 1.

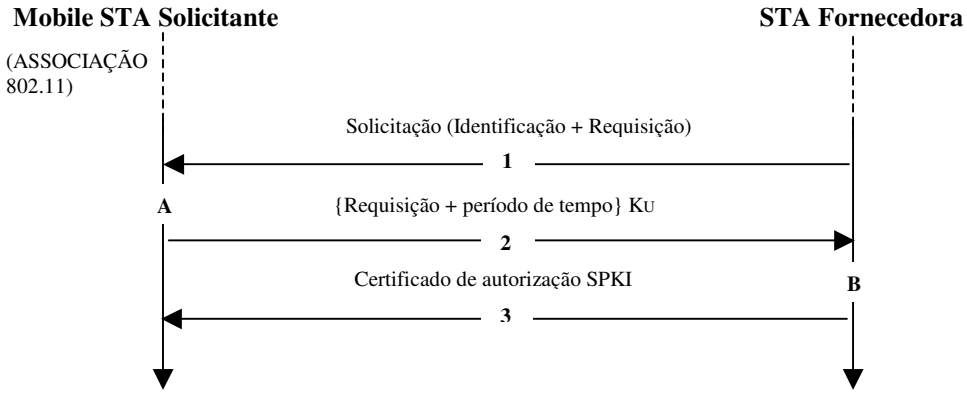

Figura 5.6 - Funcionamento do Roteiro 3.2.2

5.6. Conclusão

Neste capítulo foi apresentada em detalhes a proposta, juntamente com a descrição dos três roteiros principais (e seus sub-roteiros) que retratam o trabalho de pesquisa desenvolvido.

Através de todo o detalhamento da proposta é possível identificar a sua essência focada em sugerir melhorias para a segurança em redes IEEE 802.11, especialmente nos aspectos de autenticação/autorização, onde são utilizados certificados de autorização SPKI com o intuito de fornecer maior flexibilidade e agilidade.

Capítulo 6

Simulação da Proposta

Para complementar a definição conceitual da melhoria na infra-estrutura de segurança para redes IEEE 802.11 proposta no capítulo 5, optou-se também por construir uma simulação do seu funcionamento visando facilitar a compreensão e detectar eventuais problemas. Não existe aqui a pretensão de validar formalmente o protocolo proposto através de uma simulação, mas de permitir uma melhor observação de seu comportamento em situações próximas à realidade. A implementação da proposta sobre um ambiente real não foi possível devido a limitações técnicas e à não disponibilidade dos equipamentos adequados.

A seguir, será descrito o ambiente de simulação construído, incluindo as etapas de modelagem do problema e codificação do programa elaborado.

6.1. Descrição Detalhada

A forma escolhida para validar a proposta apresentada no capítulo 5 foi a validação informal, consistindo na criação de um programa em linguagem orientada a objetos para simular o comportamento de cada um dos elementos de uma rede *wireless* IEEE 802.11 (*peers*, APs e AS), de maneira similar ao que ocorre em uma rede *wireless* real.

A opção pela validação informal ao invés de uma validação em ambiente real foi feita por duas razões principais. A primeira foi inexistência de uma infra-estrutura *wireless* disponível (computadores portáteis, *access points* e interfaces de rede IEEE 802.11) para realizar o experimento. A segunda razão foi o fato de que, caso houvesse a infra-estrutura disponível, seria necessário que fossem realizadas alterações de código nos protocolos de

comunicação (como no TLS, por exemplo) para que suportassem a utilização de certificados SPKI, o que acarretariam um longo tempo e grande esforço pela falta de experiência em implementações de baixo nível.

Para elaborar o programa de validação, foram utilizados a UML (*Unified Modeling Language*) na fase de modelagem dos elementos e a linguagem de programação Java na fase de desenvolvimento do programa (codificação). A UML foi escolhida por tratar-se de uma linguagem para modelagem amplamente difundida e aceita internacionalmente na área de orientação a objetos. A opção pela linguagem Java foi feita em função de sua facilidade de utilização, as suas características como a possibilidade de utilização de *threads* e ao fato de ser orientada a objetos, o que facilita a representação e a implementação de objetos do mundo real.

As ferramentas utilizadas foram o software *Rose 4.0.14* da *Rational Software Corporation* para a modelagem UML e a plataforma *Eclipse 2.1.0* da *IBM Corporation* como ambiente de desenvolvimento para a linguagem Java.

A seguir, serão apresentadas detalhadamente as duas fases do desenvolvimento do programa de validação da proposta: modelagem UML e codificação em Java.

6.1.1. Modelagem UML

Os pontos de partida para a modelagem foram os três roteiros principais e seus sub-roteiros apresentados no capítulo 5. A partir deles, foram definidos inicialmente os casos de uso (*use cases*) e em seguida as classes, os relacionamentos entre elas e os seus atributos e operações (métodos).

Para simplificar a modelagem e conseqüentemente o código e o funcionamento do programa, o foco principal foi a autenticação/autorização utilizando certificados SPKI, o que implicou na não-representação de algumas mensagens do EAP-TLS.

Na definição dos casos de uso foi considerada a existência de quatro elementos distintos:

- *Manager*: é o elemento responsável por inicializar o AS, os APs e os *peers*.
- *AS*: é o elemento responsável pela autenticação/autorização dos *peers*.
- *APs*: são os elementos responsáveis por encaminhar mensagens dos *peers* para o AS e retornar as informações de processamento do AS para os *peers*.

- *Peers*: são os elementos que solicitam autenticação/autorização para poderem ter acesso aos recursos. Podem comunicar-se com os APs aos quais estejam associados ou diretamente entre si (redes *ad-hoc*).

Para a criação dos diagramas de casos de uso, foi feita uma análise nos roteiros descritos no capítulo 5 e optou-se por representá-los em quatro diagramas diferentes, utilizando os elementos anteriormente descritos:

- **Caso de uso 1**: neste diagrama estão representados os roteiros 1 (STA do usuário possui chave autorizada) e 2 (Mobile STA do usuário é capaz de gerar SPKI *certificate chain*) que foram agrupados pela sua grande semelhança de operação.

- **Caso de uso 2**: neste diagrama está representado o roteiro 3.1 (Falha na autorização do usuário) que trata-se de um sub-roteiro de 3 (*Mobile* STA do usuário não possui chave autorizada nem é capaz de gerar SPKI *chain*).

- **Caso de uso 3**: neste diagrama está representado o roteiro 3.2.1 (Busca de estações que possuam chave autorizada e poder de delegação) que também trata-se de um sub-roteiro de 3.

- **Caso de uso 4**: neste diagrama está representado o roteiro 3.2.2 (Comunicação com estação conhecida – interação pessoal prévia) que também trata-se de um sub-roteiro de 3.

A seguir, será apresentado detalhadamente cada um destes quatro casos de uso.

Caso de uso 1

Representa o processo de autenticação/autorização de um *peer* que possua chave pública autorizada ou certificados SPKI que comprovem a sua autorização. Seu primeiro passo é a inicialização dos elementos (AS, APs e *peers*), seguido pela associação do *peer* ao AP e as posteriores trocas de mensagens até que seja confirmado o sucesso na

autenticação/autorização do *peer* pelo AS, através da mensagem *EAP-Success*. A desassociação é solicitada pelo *peer* somente após encerrar a sua comunicação com o AP.

A figura a seguir demonstra a representação em UML deste caso de uso.

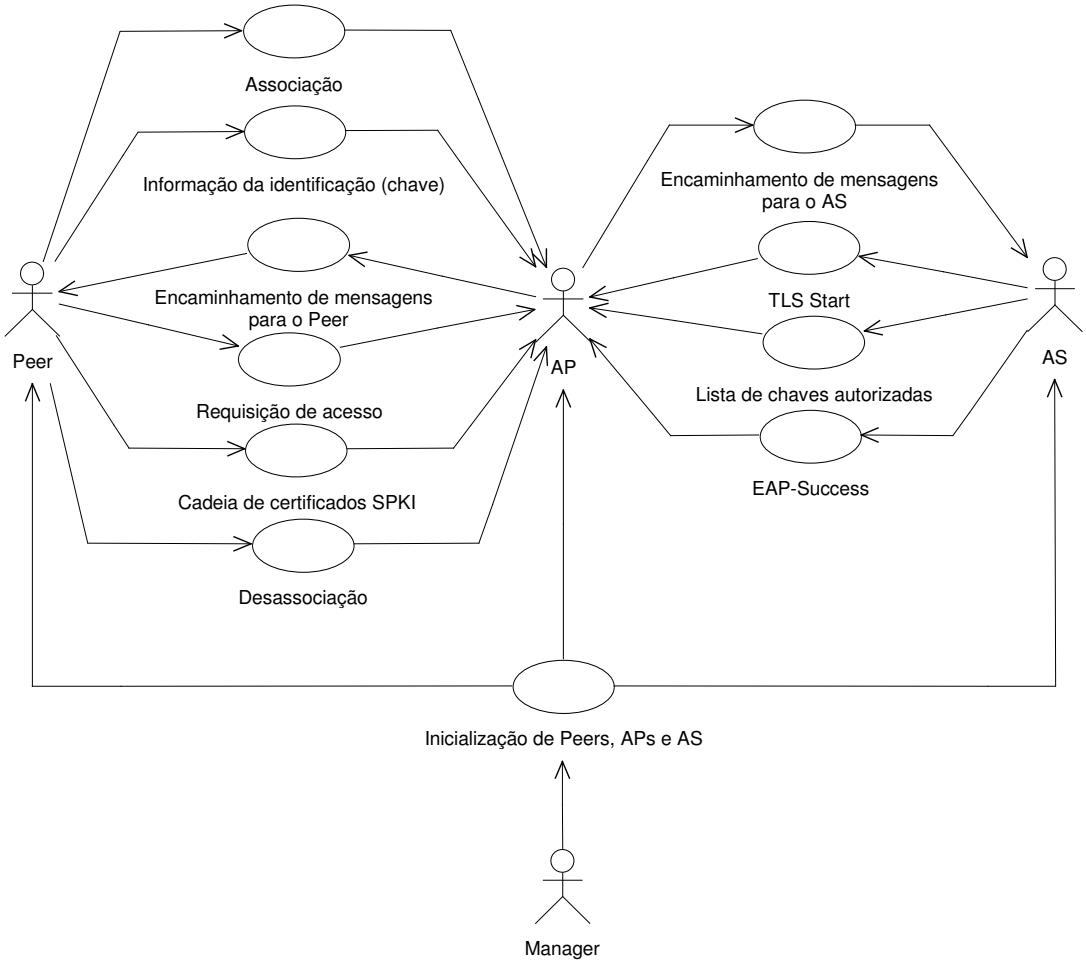

Figura 6.1 - Use case 1

Caso de uso 2

Representa o processo de falha na autenticação/autorização de um *peer* por não possuir chave pública autorizada ou certificados SPKI que comprovem a sua autorização. Seu primeiro passo é a inicialização dos elementos (AS, APs e *peers*), seguido pela associação do *peer* ao AP e as posteriores trocas de mensagens até que seja identificada falha na autenticação/autorização do *peer*, indicada através da mensagem *TLS Alert* e confirmada a não autorização do *peer* pelo AS, com a mensagem *EAP-Failure*. A desassociação é solicitada

pelo *peer* somente após encerrar a sua comunicação com o AP, que neste caso ocorre após o recebimento *de EAP-Failure*.

A figura a seguir demonstra a representação em UML deste caso de uso.

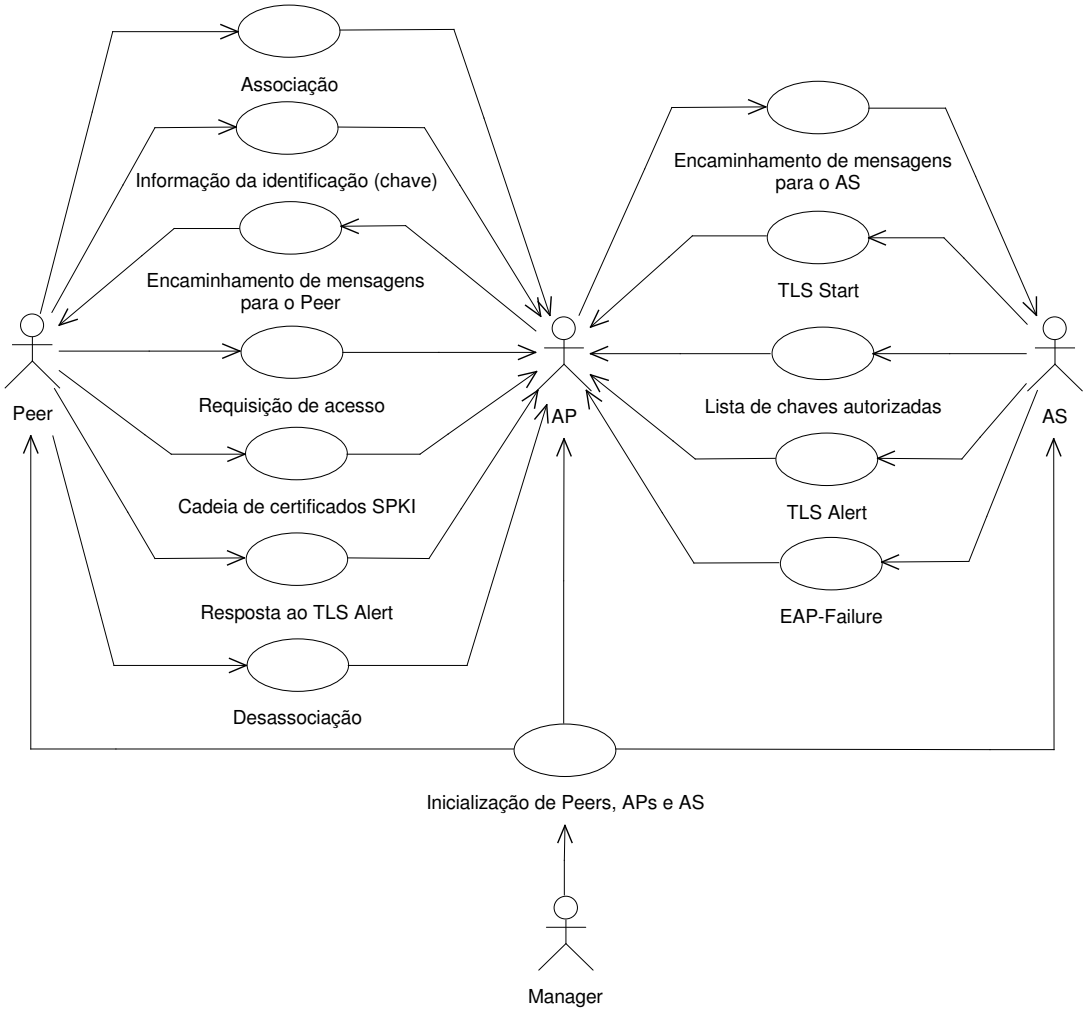

Figura 6.2 - Use case 2

Caso de uso 3

Representa o processo de busca por um *peer* fornecedor que possua chave autorizada e poder de delegação para emitir um certificado SPKI a um *peer* solicitante. Por tratar-se de uma extensão do caso de uso 2, iniciando após a desassociação do *peer* solicitante, não necessita que seja representada a etapa de inicialização dos elementos como nos casos anteriores. Seu primeiro passo é a requisição de certificado, seguido pela resposta sobre a

possibilidade de fornecimento, e posterior envio de informações necessárias para o certificado, finalizando com a sua emissão.

A figura a seguir demonstra a representação em UML deste caso de uso.

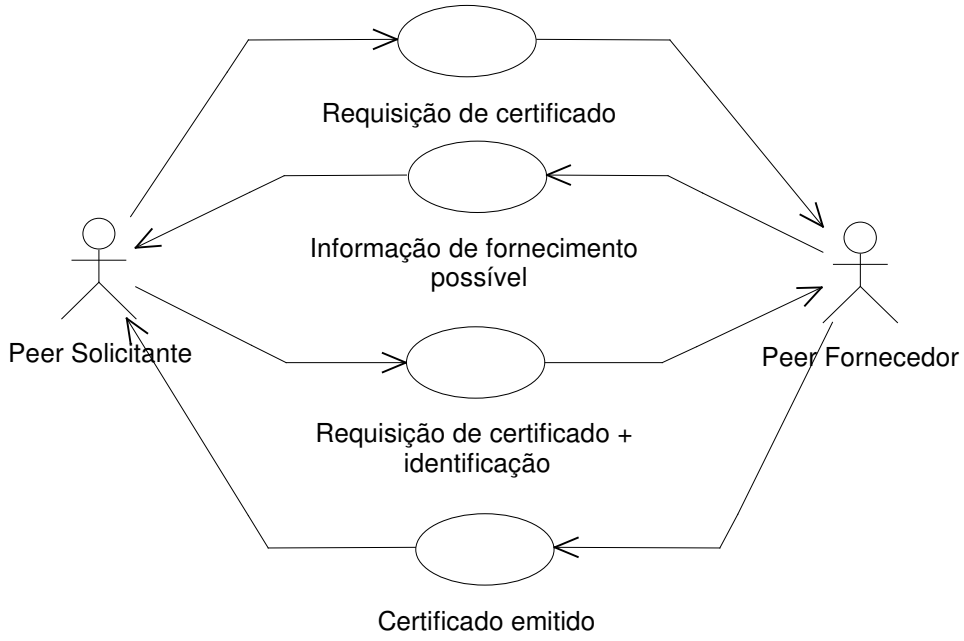

Figura 6.3 - Use Case 3

Caso de uso 4

Representa um processo similar ao caso de uso 3, diferenciando-se pela substituição da requisição inicial e informação de fornecimento por uma interação pessoal prévia entre os usuários dos *peers* solicitante e fornecedor. Por também se tratar de uma extensão do caso de uso 2, iniciando após a desassociação do *peer* solicitante, não necessita que seja representada a etapa de inicialização dos elementos. Seu primeiro passo é a consulta pessoal do usuário do *peer* solicitante ao usuário do *peer* fornecedor sobre a possibilidade de emissão de certificado, seguido pela resposta afirmativa do fornecedor. Posteriormente são solicitadas as informações necessárias para o certificado, seguidas pelo seu fornecimento, finalizando com a emissão do certificado.

A figura a seguir demonstra a representação em UML deste caso de uso.

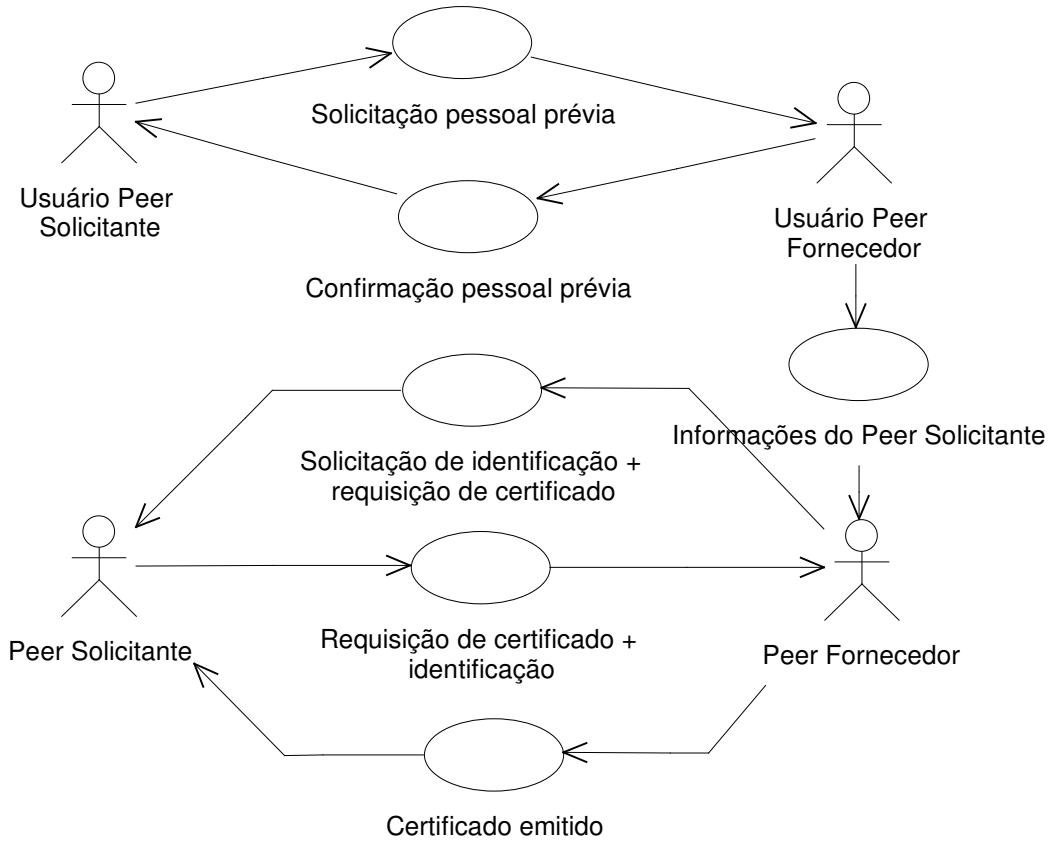

Figura 6.4 - Use Case 4

Após definidos os quatro casos de uso, foi elaborado o diagrama de classes baseando-se nas situações por eles retratadas para a identificação das classes, bem como dos atributos e operações necessários para atender as suas funcionalidades.

A seguir, será apresentado em detalhes o diagrama de classes.

Diagrama de Classes

As classes definidas a partir dos casos de uso descritos anteriormente foram as seguintes:

Manager

- Operação:
  - Inicializar_Elementos: inicializa o AS, os APs e os *peers*.

Peer

- Atributos:
  - ID: identificação do *peer*.
  - Chave_Pública (Ch_Pb): chave pública do *peer*.
  - Acesso: tipo de acesso (identificação da sub-rede) solicitado pelo *peer*.
  - ID_AP: identificação do AP ao qual o *peer* está/estará associado.
  - Certificados: certificados SPKI possuídos pelo *peer* no formato (chave do emissor, chave do beneficiário, delegação, tipo de acesso) .
- Operações:
  - Elaborar_Cadeia: elabora a cadeia de certificados SPKI do *peer*.
  - Informar_Fornecimento: analisa e informa a possibilidade de fornecimento (emissão) de um certificado solicitado.
  - Emitir_Certificado: emite certificado SPKI para o *peer* solicitante.
  - Inserir_Certificado: insere certificado emitido em sua lista de certificados.

AP (Access Point)

- Atributos:
  - ID: identificação do AP.
  - ID_Peers: identificação e chaves dos *peers* associados a ele.
- Operações:
  - Associar_Peer: associa o *peer* ao AP.
  - Registrar_Chave_Peer: registra a chave de *peer* associado ao AP.
  - Encaminhar_Identificação: encaminha a chave do *peer* para o AS e retorna indicação de início do TLS ao *peer*.
  - Encaminhar_Requisição: encaminha a requisição de acesso do *peer* para o AS e retorna lista de chaves autorizadas ao *peer*.
  - Encaminhar_Cadeia: encaminha a cadeia de certificados do *peer* para o AS e retorna o resultado do processo de autorização do *peer*.
  - Desassociar_Peer: desassocia o *peer* do AP.

AS (Authentication Server)

- Atributos:
  - ID: identificação do AS.
  - Chave_Pública (Ch_Pb): chave pública do AS.
  - Clientes_TLS: listas dos *peers* que suportam TLS.
  - Registro: lista dos registros de chaves e suas respectivas autorizações no formato (chave autorizada, tipo de acesso).
- Operações:
  - Analisar_Cliente_TLS: analisa o *peer* para identificar se ele suporta TLS.
  - Elaborar_Lista_Chaves: elabora a lista de chaves autorizadas para o tipo de autorização solicitado pelo *peer*.
  - Analisar_Autorização: analisa a chave e cadeia de certificados do *peer* para identificar se ele está autorizado.

O diagrama de classes em UML está representado a seguir:

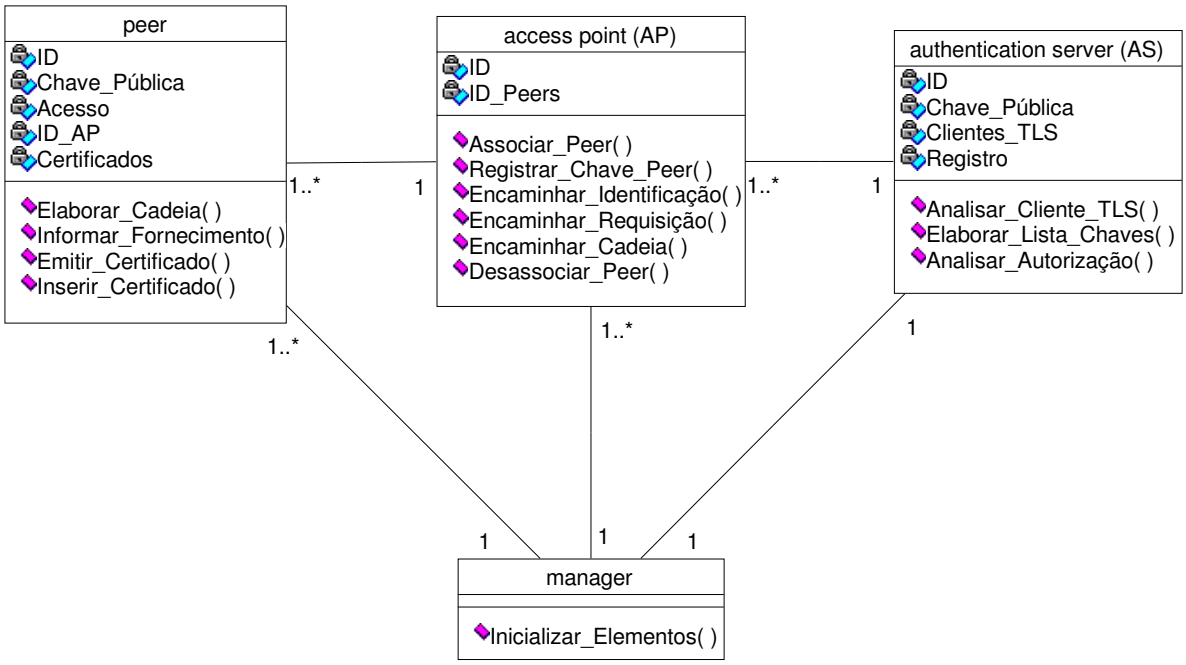

Figura 6.5 - Diagrama de Classes

6.1.2. Codificação em Java

A etapa de codificação em linguagem Java teve início a partir dos diagramas de casos de uso e de classes elaborados anteriormente. Baseando-se neles foram criados as classes, seus atributos e suas operações.

Durante a codificação foi definida a quantidade de objetos que seriam criados a partir de cada uma das três classes comunicantes, desconsiderando a classe *Manager* que tem apenas instância para inicializar as demais. Foram definidas uma instância para a classe AS (AS1), duas para a classe AP (AP1 e AP2) e quatro para a classe *Peer* (Peer1, Peer2, Peer3 e Peer4).

No caso específico da classe *Peer*, cada uma de suas instâncias (objetos) foi definida com características diferentes a fim de exemplificar os roteiros definidos no capítulo 5 e seus casos de uso relacionados. As características individuais de cada objeto *Peer* são:

- Peer1: possui uma chave pública autorizada pelo servidor de autenticação (AS1). O seu processo de autorização retrata o roteiro 1 e o caso de uso 1.
- Peer2: não possui chave pública autorizada pelo servidor nem certificado SPKI que comprove a sua autorização, o que gera uma falha de autorização do usuário, retratando o roteiro 3.1 e caso de uso 2. Posteriormente, o usuário deste peer, através de uma interação pessoal com o usuário do Peer3, obtém um certificado SPKI que comprove a sua autorização, retratando o roteiro 3.2.2 e o caso de uso 4.
- Peer3: não possui chave pública autorizada pelo servidor, mas possui um certificado SPKI que comprove a sua autorização, retratando o roteiro 2 e o caso de uso 1. Também opera como peer fornecedor para o Peer2 e o Peer4.
- Peer4: não possui chave pública autorizada pelo servidor nem certificado SPKI que comprove a sua autorização, o que acaba gerando uma falha de autorização do usuário, retratando o roteiro 3.1 e caso de uso 2. Posteriormente, este peer faz uma solicitação de emissão de certificado para o Peer3 e obtém um certificado SPKI que comprove a sua autorização, retratando o roteiro 3.2.1. e o caso de uso 3.

Para a execução do programa foi definido que seria utilizada uma máquina local, com sistema operacional Windows 2000. Entretanto, percebeu-se a necessidade de retratar com

maior fidelidade características de uma rede *wireless*, como dinamismo e comportamento não linear, o que com um código executado de maneira totalmente seqüencial seria difícil de representar.

Para resolver este problema optou-se pela definição da classe *Peer* como *thread*, possibilitando que fosse criado um ambiente *multi-thread* onde cada objeto *peer* seria executado como uma *thread* separada na JVM (*Java Virtual Machine*). Isto permite que blocos de código dos *peers* sejam executados de tal forma que, quando um *peer* entrar em estado de repouso (*sleep*), um bloco de código de outro *peer* poderá ser executado na JVM, fornecendo uma sensação de paralelismo e permitindo retratar a não linearidade do comportamento dos *peers*, como ocorre em uma rede *wireless* real. Por exemplo, o Peer2 não precisa esperar que o Peer1 finalize o seu processo de autenticação/autorização antes de iniciar sua comunicação com o AP1, podendo os dois *peers* interagirem simultaneamente com o mesmo AP.

Durante as diversas execuções do programa de simulação, percebeu-se que as interações entre os elementos (*peers*, APs e AS), apesar de não seguirem sempre a mesma sequência (conforme definido para que retratassem melhor o dinamismo de uma rede wireless), ocorreram de maneira coerente e correta, pois não demonstraram comportamento anormal de nenhum dos elemento e nem erros quanto a sequência das mensagens.

Para maiores detalhes sobre o código do programa e resultados de execução, eles estão disponíveis no Apêndice A.

6.2. Conclusão

Neste capítulo foi apresentada a descrição de como foi elaborado o programa de simulação da proposta, incluindo as etapas de modelagem e codificação. Esta simulação possibilitou a análise da proposta sob um ponto de vista prático e conseqüentemente, mais próximo de uma implementação real. Isto fornece mais subsídios para a defesa de viabilidade da utilização da proposta em um ambiente *wireless* real.

Conclusão

Nos diversos capítulos desta dissertação foram apresentados os padrões para redes móveis *wireless Bluetooth* e IEEE 802.11, protocolos e mecanismos de segurança gerais e aplicáveis a redes *wireless*, a infra-estrutura de chave pública SPKI, que caracteriza-se por utilizar certificados para conceder autorizações e possibilitar que elas sejam delegadas, e finalmente, uma proposta de melhoria para a infra-estrutura de segurança existente em redes IEEE 802.11 e sua simulação.

A proposta desenvolvida tem como objetivo fornecer uma infra-estrutura de segurança confiável para redes *wireless* IEEE 802.11 e que ao mesmo tempo proporcione flexibilidade e agilidade nos processos de autenticação e autorização, de forma a respeitar as características dinâmicas de um ambiente de redes móveis. Visando garantir confiabilidade, foram utilizados os protocolos EAP e TLS para complementar a segurança oferecida pelas camadas de mais baixo nível. Dessa forma é assegurado um bom nível de segurança durante as trocas de mensagens entre as máquinas clientes (*peers*), *os access points* e os servidores de autenticação, possibilitando a utilização de certificados para autenticação e autorização dos elementos comunicantes. A infra-estrutura SPKI foi utilizada com o objetivo de proporcionar flexibilidade e agilidade no processo de autorização através da utilização de certificados SPKI delegáveis, com a vantagem de não necessitar de uma entidade certificadora centralizada, como o que ocorre com as CAs em uma infra-estrutura de chaves públicas tradicional.

Para demonstrar a viabilidade da proposta, foi desenvolvido um programa que simulasse os comportamentos de cada um dos elementos (*peers*, *access points* e servidor de autenticação) em um ambiente *wireless* IEEE 802.11. Apesar de se tratar de uma validação informal, onde foram utilizados elementos virtuais criados e simulados em linguagem Java ao invés de máquinas reais com interfaces de rede *wireless*, foi possível observar que a utilização de certificados de autorização SPKI proporciona uma maior descentralização (eliminando a dependência de CAs para a emissão de certificados) e agilidade (possibilitando delegações) no processo de autorização.

A descentralização e agilidade ficaram demonstradas pela possibilidade de um elemento (*peer*) emitir certificados de autorização para outros (desde que o mesmo possua

uma chave autorizada ou certificados delegáveis e que as autorizações concedidas sejam iguais ou inferiores às possuídas), o que diminui a dependência de um emissor de certificados centralizador que, caso não esteja disponível, acaba impactando em todo o funcionamento da rede. Isto é possível pela igualdade existente, pois não há um elemento hierarquicamente superior, o que também permite que caso um *peer* não possa atender a uma solicitação de emissão de certificado, um outro *peer* que possua a autorização pretendida possa emiti-lo.

Entretanto, a utilização de uma validação informal acaba limitando um pouco a simulação da proposta por não possibilitar que algumas variáveis do ambiente real sejam analisadas, como o impacto das alterações nos *peers*, APs e AS para que suportem autenticação/autorização por certificados SPKI, o possível tempo adicional gasto na composição das cadeias de certificados, além do impacto da utilização de criptografia de chave pública em relação ao desempenho principalmente dos *peers*.

Apesar destas limitações, a simulação realizada conseguiu cumprir com o seu objetivo principal de demonstrar que, além de sua utilização tradicional em redes fixas, também é possível utilizar certificados SPKI para a autenticação/autorização de usuários dentro de um ambiente dinâmico, como as redes IEEE 802.11.

Como trabalhos futuros podem ser consideradas a representação e análise da proposta usando uma Técnica de Descrição Formal adequada para a representação de protocolos, como as redes de Petri. Outra possibilidade é a implementação da proposta em ambiente real, utilizando máquinas com interfaces de rede *wireless* padrão IEEE 802.11, a fim de analisar detalhadamente o comportamento de cada um dos elementos em relação a processamento e desempenho, além do impacto total das modificações necessárias para que sejam suportados certificados SPKI. Além disso, poderá ser feita uma análise da viabilidade para implementar a infra-estrutura de segurança proposta em cenários reais, como, por exemplo, aeroportos, empresas, universidades ou outros locais em que haja demanda por redes móveis temporárias que forneçam um bom nível de segurança e um processo de autenticação/autorização flexível.

Referências Bibliográficas

[1] 3COM Corporation. *IEEE 802.11b Wireless LANs: Wireless Freedom at Ethernet Speeds*. 3COM Technical Paper, 2001.

[2] Aboba, B.; Simon, D. *PPP EAP TLS Authentication Protocol*. RFC 2716. October 1999.

[3] Arbaugh, W. A.; Shankar, N. and Wang, J. *Your 802.11 Network has no Clothes*. 1st *IEEE Intl Conference on Wireless LANs and Home Networks*, December 2001.

[4] Arnold, Ken; Gosling, James. *The Java Programming Language*. Massachusetts, Addison-Wesley, 1997.

[5] Bluetooth SIG. Specification of the Bluetooth System (volume 1 – Core). Version 1.1, February 2001.

[6] Bluetooth SIG. Specification of the Bluetooth System (volume 2 – Profiles). Version 1.1, February 2001.

[7] Blunk, L.; Vollbrecht, J. *PPP Extensible Authentication Protocol (EAP).* RFC 2284. March 1998.

[8] Blunk, L.; Vollbrecht, J.; Aboba, B.; Carlson, J. *Extensible Authentication Protocol (EAP).* Internet Draft (draft-ietf-eap-rfc2284bis-00). January 2003.

[9] Borisov, N.; Goldberg, I.; Wagner, D. *Intercepting Mobile Communications: The Insecurity of 802.11*. 7th Annual International Conference on Mobile Computing And Networking, July 2001.

[10] Burnside, M.; Clarke, D.; Mills, T.; Maywah, A.; Devadas, S.; Rivest, R. *Proxy-Based Security Protocols in Networked Mobile Devices*. 17th ACM Symposium on Applied Computing (Security Track), pages 265-272, March 2002.

[11] Cisco Systems. *Cisco Aironet Wireless LAN Security Overview*. Cisco White Paper, 2002.

[12] Cisco Systems. *Extensible Authentication Protocol Transport Layer Security Deployment Guide for Wireless LAN Networks*. Cisco White Paper, 2002.

[13] Cisco Systems. *Software: Cisco Aironet and Cisco Secure Access Control Server Security Implementations for the Cisco Wireless Security Suite*. Cisco White Paper, 2002.

[14] Clarke, D. *SPKI / SDSI HTTP Server / Certificate Chain Discovery in SPKI / SDSI.* Master´s Thesis. Massachusetts Institute of Technology, 2001.

[15] Clarke, D.; Elien , J. E.; Ellison, C. ; Fredette, M.; Morcos, A. and Rivest R. L. *Certificate Chain Discovery in SPKI / SDSI*. Journal of Computer Security, 2001.

[16] Convey, S.; Miller, D. *SAFE: Wireless LAN Security in Depth*. Cisco White Paper, 2001.

[17] Dierks, T.; Allen, C. *The TLS Protocol Version 1.0.* RFC 2246. January 1999.

[18] Ellison, C. *SPKI Requirements*. RFC 2692. September 1999.

[19] Ellison, C.; Frantz, B.; Lampson, B.; Rivest R.; Thomas, B.; Ylonen, T. *SPKI Certificate Theory*. RFC 2693. September 1999.

[20] Ericsson. Bluetooth Beginners Guide. Ericsson White Paper, 2000.

[21] Fluhrer, S.; Mantin, I.; Shamir, A. *Weaknesses in the key schedule algorithm of RC4*. 4th Annual Workshop on Selected Areas of Cryptography, 2001.

[22] Furlan, José Davi. *Modelagem de Objetos através da UML*. São Paulo, Makron Books, 1998.

[23] Horstmann, C. S; Cornell, G. *Core Java Volume II – Advanced Features*. California, Sun Microsystems Press, 1998.

[24] Intel Corporation. *Wireless Security and VPN: Why VPN is Essential for Protecting Today's 802.11 Networks*. Intel Corporation White Paper, 2002.

[25] Interlink Networks. *Introduction to 802.1X for Wireless Local Area Networks*. Interlink Networks White Paper. 2002.

[26] Josefsson, S.; Andersson, H. *Protected Extensible Authentication Protocol (PEAP)*. Internet Draft (draft-josefsson-pppext-eap-tls-eap-00). August 2001.

[27] Kardach, J. *Bluetooth Architecture Overview*. Intel Technology Journal Q2, 2000.

[28] LAN MAN Standards Committee, IEEE Computer Society. *ANSI/IEEE Standard 802.11*. 1999.

[29] LAN MAN Standards Committee of the IEEE Computer Society. *Port-Based Network Access Control - IEEE Std 802.1X-2001*. IEEE, 2001.

[30] Lough, D. L.; Blankenship, T. K.; Krizman, K. J. *A Short Tutorial on Wireless LANs and IEEE 802.11*. The IEEE Computer Society's Student Newsletter, vol 5, number 2, 1997.

[31] Madhusudhana, H. S.; Ramachandran V. R. *SPKI Certificate Integration with Transport Layer Security (TLS) for Client Authentication and Authorization*. Internet Draft (draft-madhu-tls-spki-00). July 2001.

[32] Muller, T. Bluetooth Security Architecture, Version 1.0. Bluetooth SIG, July 1999.

[33] Rigney, C.; Willens, S.; Rubens, A.; Simpson, W. *Remote Authentication Dial In User Service (RADIUS)*. RFC 2865. June 2000.

[34] Roshan, P. *A Comprehensive Review of 802.11 Wireless LAN Security and the Cisco Wireless Security Suite*. Cisco White Paper, 2002.

[35] Vollbrecht, J. *Wireless LAN Access Control and Authentication: 802.11b Wireless Networking and Why It Needs Authentication*. Interlink Networks White Paper, 2002.

[36] Wang, Y. A Short Seminar on *SPKI*. Helsinki University of Technology, December 1998.

[37] Winget, N. C.; Housley, R.; Wagner, D.; Walker, J. *Security Flaws in 802.11 Data Link Protocols*. Communications of the ACM, vol. 46, nº 5, May 2003.

Apêndice A

Programa de Simulação

A.1. Código Fonte

```
/**
 * Autor: Marcello Henrique Tozoni Milanez
 * e-mails: milanez@ppgia.pucpr.br, marcellomilanez@yahoo.com
 * Data: 23/07/2003
 *
 * Copyright: O uso não-comercial (parcial ou total) deste código é permitido,
 * desde que citada sua origem. O uso comercial só poderá ser realizado com
 * autorização do autor.
 *
 * A classe AP é reponsável por:
 * - Associar peers ao AP.
 * - Registrar as chaves públicas dos peers no AP.
 * - Encaminhar as identificações dos peers para o AS.
 * - Encaminhar as requisições de acesso dos peers para o AS.
 * - Encaminhar as cadeias de certificados dos peer para o AS.
 * - Desassociar peers do AP.
 */

public class AP {
	public String ID;
	public String[][] ID_Peers = new String[4][2];

	public synchronized void Associar_Peer(Peer peer) {

		boolean peer_associado = false;
		for (int j = 0; j < 4 && peer_associado == false; j++) {
			//Se o peer já estiver associado ao AP
			if (this.ID_Peers[j][0] == peer.ID) {
				peer_associado = true;
				System.out.println(
					"O "
						+ peer.ID
						+ " já estava associado ao "
						+ this.ID
						+ " !");
			}
		}

		int i = 0;
		while (i < 4 && peer_associado == false) {
			//Se o peer não estiver associado ao AP e encontrar uma
			//posição
			//vazia na matriz de peers associados do AP
			if ((String) this.ID_Peers[i][0] == null) {
				this.ID_Peers[i][0] = peer.ID;
				peer_associado = true;
				System.out.println(
```

```
                        "O "
                                + peer.ID
                                + " foi associado com sucesso ao "
                                + this.ID
                                + " !");
                } else
                        i++;

        }

}

public synchronized String Registrar_Chave_Peer(Peer peer) {

        boolean chave_registrada = false;
        for (int j = 0; j < 4 && chave_registrada == false; j++) {
                //Se a chave do peer já estiver associada ao AP
                if (this.ID_Peers[j][1] == peer.Ch_Pb) {
                        chave_registrada = true;
                        System.out.println(
                                "A chave "
                                + peer.Ch_Pb
                                + " do "
                                + peer.ID
                                + " já estava registrada no "
                                + this.ID
                                + " na posição "
                                + j
                                + " !");
                }
        }

        int i = 0;
        while (i < 4 && chave_registrada == false) {
                //Se a chave do peer não estiver associada ao AP e encontrar uma
                //posição vazia ao lado da identificação do peer na matriz de
                //peers associados do AP
                if (this.ID_Peers[i][1] == null
                        && this.ID_Peers[i][0] == peer.ID) {
                        this.ID_Peers[i][1] = peer.Ch_Pb;
                        chave_registrada = true;
                        System.out.println(
                                "A chave "
                                + peer.Ch_Pb
                                + " do "
                                + peer.ID
                                + " foi registrada com sucesso no "
                                + this.ID
                                + " !");

                } else
                        i++;

        }
        return peer.Ch_Pb;
}

public synchronized boolean Encaminhar_Identificacao(
        String id_peer,
        Peer peer,
        AS as) {

        boolean cliente_tls;
        //Encaminhamento da chave do Peer registrada no AP para o AS1,
        //retornando do AS1 se o peer suporta TLS
        System.out.println(
                "A identificação (chave pública) "
                        + id_peer
                        + " do peer foi encaminhada do "
                        + this.ID
                        + " para o AS1 !");
        cliente_tls = as.Analisar_Cliente_TLS(id_peer, peer, this);

        //Envio da mensagem indicando início do TLS do AP para o peer
```

```
		if (cliente_tls == true) {
			System.out.println(
				"A mensagem 'TLS Start' foi encaminhada pelo "
					+ this.ID
					+ " para o "
					+ peer.ID
					+ "!");
		} else {
			System.out.println("O " + peer.ID + " não suporta TLS !");
		}
		return cliente_tls;
	}

	public synchronized String[] Encaminhar_Requisicao(Peer peer, AS as) {

		//Encaminhamento da requisição de acesso do AP para o AS1
		//retornando do AS1 a lista de chaves autorizadas
		System.out.println(
			"A requisição de acesso do tipo '"
				+ peer.Acesso
				+ "' criada pelo "
				+ peer.ID
				+ " foi encaminhada pelo "
				+ peer.ID_AP
				+ " para o AS1 !");
		String[] lista_chaves = new String[5];
		lista_chaves = as.Elaborar_Lista_Chaves(peer.Acesso, peer);

		//Encaminhamento da lista de chaves autorizadas pelo AP para o peer
		System.out.println(
			"A lista contendo as chaves autorizadas '"
				+ lista_chaves[0]
				+ " "
				+ lista_chaves[1]
				+ " "
				+ lista_chaves[2]
				+ "' foi encaminhada pelo "
				+ peer.ID_AP
				+ " para o "
				+ peer.ID
				+ " !");
		return lista_chaves;
	}

	public synchronized boolean Encaminhar_Cadeia(
		String[][] cadeia_peer,
		String[] chaves_aut,
		Peer peer,
		AS as) {

		boolean peer_aut = false;
		//Encaminhamento da mensagem (cadeia de certificados + requisição +
		//chave pública peer) do AP para o AS1 retornando o resultado do
		//processo de autorização do peer
		System.out.println(
			"A cadeia de certificados '"
				+ cadeia_peer[0][0]
				+ " "
				+ cadeia_peer[0][1]
				+ " "
				+ cadeia_peer[0][2]
				+ " "
				+ cadeia_peer[0][3]
				+ "' do "
				+ peer.ID
				+ " foi encaminhada pelo "
				+ peer.ID_AP
				+ " para o AS1 juntamente com a requisição '"
				+ peer.Acesso
				+ "' e a chave pública do peer '"
				+ peer.Ch_Pb
				+ "'!");
		peer_aut = as.Analisar_Autorizacao(cadeia_peer, peer, chaves_aut);
```

```
		//Se o processo de autorização do peer foi concluído com sucesso
		if (peer_aut == true) {
			//O AP1 encaminha a mensagem 'EAP-Success' para o Peer1
			System.out.println(
				"A mensagem 'EAP-Success' foi encaminhada pelo "
					+ peer.ID_AP
					+ " para o "
					+ peer.ID
					+ ", indicando que o processo de autorização do "
					+ peer.ID
					+ " foi concluído com SUCESSO !");
		}
		//Caso contrário
		else {
			//O AP encaminha a mensagem 'TLS Alert' para o Peer
			System.out.println(
				"A mensagem 'TLS Alert' foi encaminhada pelo "
					+ peer.ID_AP
					+ " para o "
					+ peer.ID
					+ ", indicando que o houve falha no processo de "
					+ "autorização do "
					+ peer.ID
					+ " !");

		}

		return peer_aut;
	}

	public synchronized void Desassociar_Peer(Peer peer) {

		boolean peer_desassociado = false;

		int i = 0;
		while (i < 4 && peer_desassociado == false) {
			//Se encontrar a identificação do peer na matriz de peers
			//associados do AP, então apagar a identificação e a
			//chave do respectivo peer
			if ((String) this.ID_Peers[i][0] == peer.ID) {
				this.ID_Peers[i][0] = null;
				this.ID_Peers[i][1] = null;
				peer_desassociado = true;
				System.out.println(
					"O "
						+ peer.ID
						+ " foi desassociado com sucesso do "
						+ this.ID
						+ " !");

			} else {
				i++;
			}

		}
	}

}




/**
 * Autor: Marcello Henrique Tozoni Milanez
 * e-mails: milanez@ppgia.pucpr.br, marcellomilanez@yahoo.com
 * Data: 23/07/2003
 *
 * Copyright: O uso não-comercial (parcial ou total) deste código é permitido,
 * desde que citada sua origem. O uso comercial só poderá ser realizado com
 * autorização do autor.
 *
 * A classe AS é responsável por:
 * - Analisar os peers para identificar se são clientes TLS.
```

```
 * - Elaborar lista de chaves autorizadas para os peer.
 * - Analisar a autorização dos peers.
 */

public class AS {
	public String ID;
	public String Ch_Pb;
	public String[] Clientes_TLS = new String[4];
	public String[][] registro = new String[5][2];

	public synchronized boolean Analisar_Cliente_TLS(
		String chave_peer,
		Peer peer,
		AP ap) {

		boolean cliente_tls = false;
		for (int i = 0; i < 4 && cliente_tls == false; i++) {
			//Se a chave do peer estiver presente na lista de clientes que
			//suportam TLS
			if (this.Clientes_TLS[i] == chave_peer) {
				cliente_tls = true;
			}

		}
		//Se o peer suportar TLS
		if (cliente_tls == true) {
			System.out.println(
				"A mensagem 'TLS Start' destinada ao "
					+ peer.ID
					+ " foi enviada pelo "
					+ this.ID
					+ " para o "
					+ ap.ID
					+ " !");
		}
		//Se o peer não suportar TLS
		else {
			System.out.println("O " + peer.ID + " não suporta TLS !");
		}

		return cliente_tls;
	}

	public synchronized String[] Elaborar_Lista_Chaves(
		String acesso_peer,
		Peer peer) {

		String[] lista = new String[5];
		int j = 0;
		for (int i = 0; i < 5 && j < 5; i++) {
			//Se o tipo de acesso contido no registro do AS for igual ao
			//solicitado pelo peer, então armazena na lista a chave
			//autorizada correspondente
			if (this.registro[i][1] == acesso_peer && lista[j] == null) {
				lista[j] = this.registro[i][0];
				j++;
			}

		}

		//Envio da lista de chaves autorizadas pelo AS1 para o AP
		System.out.println(
			"A lista contendo as chaves autorizadas '"
				+ lista[0]
				+ " "
				+ lista[1]
				+ " "
				+ lista[2]
				+ "' destinada ao "
				+ peer.ID
				+ " foi enviada pelo AS1 para o "
				+ peer.ID_AP
				+ " !");
		return lista;
```

```
}

public synchronized boolean Analisar_Autorizacao(
	String[][] cadeia,
	Peer peer,
	String[] chaves_aut) {

	boolean autorizado = false;
	String tipo_acesso = peer.Acesso;
	String chave_peer = peer.Ch_Pb;

	for (int j = 0; j < 5 && autorizado == false; j++) {
		//Se o peer possui uma chave contida na lista de chaves
		//autorizadas, então ele está autorizado
		if (chaves_aut[j] == chave_peer)
			autorizado = true;
	}

	//Caso o peer não possua chave autorizada, a sua cadeia de
	//certificados é analisada pelo AS1
	for (int i = 0; i < 2 && autorizado == false; i++) {

		for (int j = 0; j < 5; j++) {
			//Se o emissor do primeiro certificado da cadeia do
			//peer for uma chave autorizada, o peer for
			//beneficiário e o acesso solicitado pelo peer for
			//igual ao contido no certificado, então o peer
			//está autorizado
			if (cadeia[i][0] == chaves_aut[j]
				&& cadeia[i][1] == chave_peer
				&& cadeia[i][3] == tipo_acesso) {
				autorizado = true;

			} else {
				//Se o peer não for o beneficiário no certificado
				//emitido
				//por chave autorizada, mas tiver a autorização
				//concedida
				//(delegada)por este beneficiário em um outro
				//certificado
				//então o peer está autorizado
				if (cadeia[i][0] == chaves_aut[j]
					&& cadeia[i][1] == cadeia[i + 1][0]
					&& cadeia[i + 1][1] == chave_peer
					&& cadeia[i][3] == tipo_acesso
					&& cadeia[i][2] == "verdadeiro") {
					autorizado = true;

				}
			}
		}
	}
	//Se a autorização do peer for bem sucedida
	if (autorizado == true) {
		//O AS1 envia uma mensagem 'EAP-Success' para o AP
		System.out.println(
			"A mensagem 'EAP-Success' foi enviada pelo AS1 para o "
				+ peer.ID_AP
				+ ", indicando que o processo de autorização do "
				+ peer.ID
				+ " foi concluído com SUCESSO !");
	}
	//Se a autorização do peer NÃO for bem sucedida
	else {
		//O AS1 envia uma mensagem 'TLS Alert' para o AP
		System.out.println(
			"A mensagem 'TLS Alert' foi enviada pelo AS1 para o "
				+ peer.ID_AP
				+ ", indicando que houve falha no processo de"
				+ " autorização do "
				+ peer.ID
				+ " !");
	}
	return autorizado;
```

```
	}

}



/**
 * Autor: Marcello Henrique Tozoni Milanez
 * e-mails: milanez@ppgia.pucpr.br, marcellomilanez@yahoo.com
 * Data: 23/07/2003
 *
 * Copyright: O uso não-comercial (parcial ou total) deste código é permitido,
 * desde que citada sua origem. O uso comercial só poderá ser realizado com
 * autorização do autor.
 *
 * A classe Peer é responsável por:
 * - Informar sua chave pública ao AP.
 * - Informar o tipo de autorização desejada ao AP.
 * - Elaborar cadeia de certificados.
 * - Informar viabilidade de fornecimento de certificado a outro peer.
 * - Emitir certificado a outro peer.
 * - Inserir o certificado obtido em sua lista.
 */

public class Peer extends Thread {

	public String ID;
	public String Ch_Pb;
	public String Acesso;
	public String ID_AP;
	public String[][] cert = new String[3][4];

	private Peer peer_forn;
	private AP ap;
	private AS as;

	public Peer(AP ac_pn, AS at_sr, Peer pe_frn) {
		ap = ac_pn;
		as = at_sr;
		peer_forn = pe_frn;
	}

	public String[][] Elaborar_Cadeia(String[] chaves_aut) {

		String[][] cadeia = new String[3][4];
		boolean peer_autorizado = false;
		//Verifica se o peer possui chave autorizada pelo servidor
		for (int ind = 0; ind < 3 && peer_autorizado == false; ind++) {
			if (this.Ch_Pb == chaves_aut[ind])
				peer_autorizado = true;
		}

		//Caso o peer não possua chave autorizada a cadeia de certificados é
		//elaborada
		for (int i = 0; i < 3 && peer_autorizado == false; i++) {

			for (int j = 0; j < 3; j++) {
				//Se o emissor do certificado possuído pelo peer for uma
				//chave
				//autorizada e o peer for beneficiário
				if (this.cert[i][0] == chaves_aut[j]
					&& this.cert[i][1] == this.Ch_Pb) {
					peer_autorizado = true;
					cadeia[0][0] = this.cert[i][0];
					cadeia[0][1] = this.cert[i][1];
					cadeia[0][2] = this.cert[i][2];
					cadeia[0][3] = this.cert[i][3];
				} else {
					//Se o beneficiário no certificado emitido por
					//chave
					//autorizada não for o peer, mas se o peer tiver a
					//autorização concedida por este beneficiário em um
					//outro certificado
```

```
                        if (this.cert[i][0] == chaves_aut[j]
                              && this.cert[i][1] == this.cert[i + 1][0]
                              && this.cert[i + 1][1] == this.Ch_Pb) {
                              peer_autorizado = true;
                              cadeia[0][0] = this.cert[i][0];
                              cadeia[0][1] = this.cert[i][1];
                              cadeia[0][2] = this.cert[i][2];
                              cadeia[0][3] = this.cert[i][3];
                              cadeia[1][0] = this.cert[i + 1][0];
                              cadeia[1][1] = this.cert[i + 1][1];
                              cadeia[1][2] = this.cert[i + 1][2];
                              cadeia[1][3] = this.cert[i + 1][3];
                        }
                  }
            }
      }

      return cadeia;

}

public boolean Informar_Fornecimento(
      String[] lista_chaves,
      String tipo_aut) {

      boolean fornecimento_possivel;
      fornecimento_possivel = false;
      for (int ind = 0; ind < 5 && fornecimento_possivel == false; ind++) {
            //Se o peer fornecedor possuir uma chave autorizada
            //poderá criar um certificado para o solicitante
            if (this.Ch_Pb == lista_chaves[ind]) {
                  fornecimento_possivel = true;
            }
      }

      for (int i = 0; i < 3 && fornecimento_possivel == false; i++) {

            for (int j = 0; j < 5 && fornecimento_possivel == false; j++) {
                  //Se o emissor de um dos certificados do peer fornecedor
                  //for
                  //uma chave autorizada e se o beneficiário deste
                  //certificado
                  //for o peer fornecedor e se este certificado tiver a
                  //autorização pedida pelo peer solicitante e se este
                  //certificado for delegável, o fornecedor poderá criar um
                  //certificado para o solicitante

                  if (this.cert[i][0] == lista_chaves[j]
                        && this.cert[i][1] == this.Ch_Pb
                        && this.cert[i][3] == tipo_aut
                        && this.cert[i][2] == "verdadeiro") {
                        fornecimento_possivel = true;
                  }
            }
      }

      return fornecimento_possivel;
}

public String[][] Emitir_Certificado(
      String[] lista_chaves,
      String tipo_aut,
      String chave_peer) {

      String[][] certificado = new String[2][4];
      boolean cert_emitido = false;

      for (int ind = 0; ind < 5 && cert_emitido == false; ind++) {
            //Se o peer fornecedor possuir uma chave autorizada
            //criará um certificado para o solicitante
            if (this.Ch_Pb == lista_chaves[ind]) {
                  certificado[0][0] = this.Ch_Pb;
                  certificado[0][1] = chave_peer;
                  certificado[0][2] = "falso";
```

```
                        certificado[0][3] = tipo_aut;
                        cert_emitido = true;

                }
        }

        for (int i = 0; i < 3 && cert_emitido == false; i++) {
                for (int j = 0; j < 5 && cert_emitido == false; j++) {
                        //Se o emissor de um dos certificados do peer fornecedor
                        //for
                        //uma chave autorizada e se o beneficiário deste
                        //certificado
                        //for o peer fornecedor e se este certificado tiver a
                        //autorização pedida pelo peer solicitante e se este
                        //certificado for delegável, o fornecedor criará um
                        //certificado para o solicitante
                        if (this.cert[i][0] == lista_chaves[j]
                                && this.cert[i][1] == this.Ch_Pb
                                && this.cert[i][3] == tipo_aut
                                && this.cert[i][2] == "verdadeiro") {
                                certificado[0][0] = this.cert[i][0];
                                certificado[0][1] = this.cert[i][1];
                                certificado[0][2] = this.cert[i][2];
                                certificado[0][3] = this.cert[i][3];
                                certificado[1][0] = this.Ch_Pb;
                                certificado[1][1] = chave_peer;
                                certificado[1][2] = "falso";
                                certificado[1][3] = tipo_aut;
                                cert_emitido = true;

                        }
                }
        }

        return certificado;
}

public void Inserir_Certificado(String[][] certificados) {
        boolean cert_inserido = false;

        for (int i = 0; i < 3 && cert_inserido == false; i++) {
                //Se houver uma posição vazia na matriz de certificados do peer,
                //então ele insere o certificado
                if (this.cert[i][0] == null) {
                        for (int j = 0; j < 4; j++) {
                                this.cert[i][j] = certificados[0][j];

                        }
                        cert_inserido = true;
                        //Se houver mais um certificado a ser inserido e uma
                        // posição
                        //vazia na matriz de certificados do peer, então insere-o
                        if (this.cert[i + 1][0] == null
                                && certificados[1][0] != null) {
                                for (int k = 0; k < 4; k++) {
                                        this.cert[i + 1][k] = certificados[1][k];

                                }
                        }
                }

        }
}

public void run() {

        //Associação do Peer ao AP
        ap.Associar_Peer(this);

        //Envio da identificação (chave pública) do peer para o AP
        System.out.println(
                "A identificação (chave pública) "
                        + this.Ch_Pb
                        + " do peer foi enviada pelo "
```

```
                + this.ID
                + " para o "
                + this.ID_AP
                + " !");

//Registro da chave pública do Peer no AP
String chave_peer;
chave_peer = ap.Registrar_Chave_Peer(this);

//Encaminhamento da identificação (chave pública) do peer para o AS1
//realizado pelo AP e retorno da correta identificação do peer
//pelo AS1
boolean id_peer_ok = false;
id_peer_ok = ap.Encaminhar_Identificacao(chave_peer, this, as);

//Faz a thread em execução entrar em repouso para permitir que
//outra thread possa ser executada
for (int i = 0; i < 10; i++) {
      try {
            Thread.sleep(1);
            if (i == 0) {
                  System.out.println(
                        "***Início do repouso da thread '"
                        + this.ID + "' !");
            }
            if (i == 9) {
                  System.out.println(
                        "*****Fim do repouso da thread '"
                        + this.ID + "' !");
            }

      } catch (InterruptedException e) {
            return;
      }
}

//Envio da requisição de acesso do peer para o AP
System.out.println(
      "A requisição de acesso do tipo '"
            + this.Acesso
            + "' foi enviada pelo "
            + this.ID
            + " para o "
            + this.ID_AP
            + " !");

//Encaminhamento da requisição de acesso do peer para o AS1 realizado
//pelo AP e retorno da lista de chaves autorizadas elaborada pelo AS1
String[] chaves_aut = new String[5];
chaves_aut = ap.Encaminhar_Requisicao(this, as);

//O peer elabora a sua cadeia de certificados
String[][] cadeia_peer = new String[3][4];
cadeia_peer = this.Elaborar_Cadeia(chaves_aut);

//Envio da mensagem (cadeia de certificados + requisição + chave
//pública peer) do peer para o AP
System.out.println(
      "A cadeia de certificados '"
            + cadeia_peer[0][0]
            + " "
            + cadeia_peer[0][1]
            + " "
            + cadeia_peer[0][2]
            + " "
            + cadeia_peer[0][3]
            + "' foi enviada pelo "
            + this.ID
            + " para o "
            + this.ID_AP
            + " juntamente com a requisição '"
            + this.Acesso
            + "' e a chave pública do peer '"
            + this.Ch_Pb
```

```
                + "'!");

        //Encaminhamento da mensagem (cadeia de certificados + requisição +
        //chave pública peer) do peer para o AS1 realizado pelo AP e
        //retorno da correta ou incorreta autenticação do peer pelo AS1
        boolean peer_autorizado = false;
        peer_autorizado =
               ap.Encaminhar_Cadeia(cadeia_peer, chaves_aut, this, as);

        //Se o processo de autorização do peer não foi concluído com sucesso
        if (peer_autorizado == false) {
               //O Peer, neste caso, por não possuir comprovação de autorização
               //envia um pacote VAZIO para o AP
               System.out.println(
                      "Um pacote vazio foi enviado pelo "
                             + this.ID
                             + " para o "
                             + this.ID_AP
                             + " em resposta ao 'TLS Alert'!");
               //O AP encaminha o pacote VAZIO para o AS1
               System.out.println(
                      "O pacote vazio do "
                             + this.ID
                             + " foi encaminhado pelo "
                             + this.ID_AP
                             + " para o AS1 em resposta ao 'TLS Alert'!");
               //O AS1 envia uma mensagem 'EAP-Failure' ao AP indicando que o
               //Peer NÃO conseguiu comprovar autorização
               System.out.println(
                      "A mensagem 'EAP-Failure' foi enviada pelo AS1 para o "
                             + this.ID_AP
                             + ", indicando que o processo de autorização do "
                             + this.ID
                             + " NÃO foi concluído com sucesso!");
               //O AP encaminha a mensagem 'EAP-Failure' para o Peer
               System.out.println(
                      "A mensagem 'EAP-Failure' foi encaminhada pelo "
                             + this.ID_AP
                             + " para o "
                             + this.ID
                             + ", indicando que o processo de autorização do "
                             + this.ID
                             + " NÃO foi concluído com sucesso!");
               //O Peer desassocia-se do AP
               ap.Desassociar_Peer(this);
        }

        if (this.ID == "Peer4") {

               //O Peer4(solicitante) envia uma requisição para emissão de
               //certificado de autorização SPKI para o Peer3(fornecedor)
               System.out.println(
                      "A requisição para emissão de certificado de autorização"
                             + " foi enviada pelo "
                             + this.ID
                             + " para o "
                             + peer_forn.ID
                             + " !");

               //O Peer3 analisa a requisição do Peer4 e define se poderá
               //emitir
               //o certificado conforme solicitado
               boolean fornec_possivel;
               fornec_possivel =
                      peer_forn.Informar_Fornecimento(chaves_aut, this.Acesso);

               if (fornec_possivel == true) {
                      System.out.println(
                             "A informação 'Emissão de certificado possível'"
                                    + " foi enviada do "
                                    + peer_forn.ID
                                    + " para o "
                                    + this.ID
                                    + " !");
```

```
} else {
	System.out.println(
		"A informação 'Emissão de certificado NÃO "
			+ " possível' foi enviada do "
			+ peer_forn.ID
			+ " para o "
			+ this.ID
			+ " !");
}

//O Peer4 envia sua requisição de emissão de certificado de
//autorização juntamente com sua chave para o Peer3
System.out.println(
	"A requisição para a emissão de certificado de "
		+ " autorização + chave foi enviada pelo "
		+ this.ID
		+ " para o "
		+ peer_forn.ID
		+ " !");

//O Peer3 emite o certificado de autorização para o Peer4
String[][] cert_emitido = new String[2][4];
cert_emitido =
	peer_forn.Emitir_Certificado(
		chaves_aut,
		this.Acesso,
		this.Ch_Pb);
System.out.println(
	"Os certificados: '"
		+ cert_emitido[0][0]
		+ " "
		+ cert_emitido[0][1]
		+ " "
		+ cert_emitido[0][2]
		+ " "
		+ cert_emitido[0][3]
		+ "' e  '"
		+ cert_emitido[1][0]
		+ " "
		+ cert_emitido[1][1]
		+ " "
		+ cert_emitido[1][2]
		+ " "
		+ cert_emitido[1][3]
		+ "' foram enviados pelo "
		+ peer_forn.ID
		+ " para o "
		+ this.ID
		+ " !");

//O Peer4 insere o(s) certificado(s) recebidos na sua matriz de
//certificados
this.Inserir_Certificado(cert_emitido);
System.out.println(
	"Os certificados: '"
		+ this.cert[1][0]
		+ " "
		+ this.cert[1][1]
		+ " "
		+ this.cert[1][2]
		+ " "
		+ this.cert[1][3]
		+ "' e  '"
		+ this.cert[2][0]
		+ " "
		+ this.cert[2][1]
		+ " "
		+ this.cert[2][2]
		+ " "
		+ this.cert[2][3]
		+ "' foram inseridos na matriz de certificados do "
		+ this.ID
		+ " !");
```

```
}

if (this.ID == "Peer2") {
	//O usuário do Peer2(solicitante) realiza uma consulta pessoal
	//prévia ao usuário do Peer3(fornecedor)
	System.out.println(
		"O usuário do "
			+ this.ID
			+ " faz uma consulta pessoal ao usuário do "
			+ peer_forn.ID
			+ " sobre a possibilidade de emitir um certificado"
			+ " com a autorização '"
			+ this.Acesso
			+ "' !");

	//O usuário do Peer3 analisa a possibilidade de atendimento da
	//solicitação e responde pessoalmente ao usuário do Peer2
	System.out.println(
		"O usuário do "
			+ peer_forn.ID
			+ " analisa o que foi solicitado, identifica que o"
			+ " peer pode atende-la e responde ao usuário do "
			+ this.ID
			+ " !");

	//O Peer3 solicita ao Peer2 informações para a emissão do
	//certificado
	System.out.println(
		"O "
			+ peer_forn.ID
			+ " solicita ao "
			+ this.ID
			+ " a sua identificação (chave pública) e a "
			+ "requisição"
			+ "(lista de chaves autorizadas e tipo de acesso) "
			+ "necessários para a emissão do certificado !");

	//O Peer2 envia sua requisição de emissão de certificado de
	//autorização juntamente com sua chave para o Peer3
	System.out.println(
		"A requisição para a emissão de certificado de "
			+ " autorização + chave foi enviada pelo "
			+ this.ID
			+ " para o "
			+ peer_forn.ID
			+ " !");

	//O Peer3 emite o certificado de autorização para o Peer2
	String[][] cert_emitido = new String[2][4];
	cert_emitido =
		peer_forn.Emitir_Certificado(
			chaves_aut,
			this.Acesso,
			this.Ch_Pb);
	System.out.println(
		"Os certificados: '"
			+ cert_emitido[0][0]
			+ " "
			+ cert_emitido[0][1]
			+ " "
			+ cert_emitido[0][2]
			+ " "
			+ cert_emitido[0][3]
			+ "' e  '"
			+ cert_emitido[1][0]
			+ " "
			+ cert_emitido[1][1]
			+ " "
			+ cert_emitido[1][2]
			+ " "
			+ cert_emitido[1][3]
			+ "' foram enviados pelo "
			+ peer_forn.ID
			+ " para o "
```

```
                                + this.ID
                                + " !");

                        //O Peer2 insere o(s) certificado(s) recebidos na sua matriz de
                        //certificados
                        this.Inserir_Certificado(cert_emitido);
                        System.out.println(
                              "Os certificados: '"
                                + this.cert[1][0]
                                + " "
                                + this.cert[1][1]
                                + " "
                                + this.cert[1][2]
                                + " "
                                + this.cert[1][3]
                                + "' e  '"
                                + this.cert[2][0]
                                + " "
                                + this.cert[2][1]
                                + " "
                                + this.cert[2][2]
                                + " "
                                + this.cert[2][3]
                                + "' foram inseridos na matriz de certificados do "
                                + this.ID
                                + " !");

                  }

            }
}



/**
 * Autor: Marcello Henrique Tozoni Milanez
 * e-mails: milanez@ppgia.pucpr.br, marcellomilanez@yahoo.com
 * Data: 23/07/2003
 *
 * Copyright: O uso não-comercial (parcial ou total) deste código é permitido,
 * desde que citada sua origem. O uso comercial só poderá ser realizado com
 * autorização do autor.
 *
 * A classe Manager é responsável por:
 * - Inicializar o AS, os APs e os Peers.
 */

public class Manager {

      public static void main(String[] args) {

            //Inicialização do AS1
            AS as1 = new AS();
            as1.ID = "AS1";
            as1.Ch_Pb = "Kas1";
            as1.Clientes_TLS[0] = "Kp1";
            as1.Clientes_TLS[1] = "Kp2";
            as1.Clientes_TLS[2] = "Kp3";
            as1.Clientes_TLS[3] = "Kp4";
            System.out.println(
                  "Servidor de autenticação '"
                        + as1.ID
                        + "' inicializado com a chave '"
                        + as1.Ch_Pb
                        + "' !");

            //Criação dos registros de autorização do AS1
            //Registro de autorização 1
            as1.registro[0][0] = "Kas";
            as1.registro[0][1] = "acesso sub-rede I";
            System.out.println(
                  "Registro de autorização 1: '"
                        + as1.registro[0][0]
```

```
                + ", "
                + as1.registro[0][1]
                + "' criado !");

//Registro de autorização 2
as1.registro[1][0] = "Kas1";
as1.registro[1][1] = "acesso sub-rede I";
System.out.println(
       "Registro de autorização 2: '"
                + as1.registro[1][0]
                + ", "
                + as1.registro[1][1]
                + "' criado !");

//Registro de autorização 3
as1.registro[2][0] = "Kas2";
as1.registro[2][1] = "acesso sub-rede A";
System.out.println(
       "Registro de autorização 3: '"
                + as1.registro[2][0]
                + ", "
                + as1.registro[2][1]
                + "' criado !");

//Registro de autorização 4
as1.registro[3][0] = "Kas3";
as1.registro[3][1] = "acesso sub-rede A";
System.out.println(
       "Registro de autorização 4: '"
                + as1.registro[3][0]
                + ", "
                + as1.registro[3][1]
                + "' criado !");

//Registro de autorização 5
as1.registro[4][0] = "Kp1";
as1.registro[4][1] = "acesso sub-rede I";
System.out.println(
       "Registro de autorização 5: '"
                + as1.registro[4][0]
                + ", "
                + as1.registro[4][1]
                + "' criado !");

//Inicialização do AP1
AP ap1 = new AP();
ap1.ID = "AP1";
ap1.ID_Peers[0][0] = "Peer1";
System.out.println("AP '" + ap1.ID + "' inicializado !");

//Inicialização do AP2
AP ap2 = new AP();
ap2.ID = "AP2";
System.out.println("AP '" + ap2.ID + "' inicializado !");

//Inicialização do Peer1
Peer peer1 = new Peer(ap1, as1, null);
peer1.ID = "Peer1";
peer1.Ch_Pb = "Kp1";
peer1.Acesso = "acesso sub-rede I";
peer1.ID_AP = "AP1";
peer1.cert[0][0] = "Kas2";
peer1.cert[0][1] = "Kp1";
peer1.cert[0][2] = "falso";
peer1.cert[0][3] = "acesso sub-rede A";
System.out.println(
       "O peer '"
                + peer1.ID
                + "' foi inicializado com a chave '"
                + peer1.Ch_Pb
                + "' e o certificado '"
                + peer1.cert[0][0]
                + ", "
                + peer1.cert[0][1]
```

```
                + ", "
                + peer1.cert[0][2]
                + ", "
                + peer1.cert[0][3]
                + "' !");

//Inicialização do Peer3
Peer peer3 = new Peer(ap2, as1, null);
peer3.ID = "Peer3";
peer3.Ch_Pb = "Kp3";
peer3.Acesso = "acesso sub-rede I";
peer3.ID_AP = "AP2";
peer3.cert[0][0] = "Kas";
peer3.cert[0][1] = "Kp3";
peer3.cert[0][2] = "verdadeiro";
peer3.cert[0][3] = "acesso sub-rede I";
System.out.println(
       "O peer '"
                + peer3.ID
                + "' foi inicializado com a chave '"
                + peer3.Ch_Pb
                + "' e o certificado '"
                + peer3.cert[0][0]
                + ", "
                + peer3.cert[0][1]
                + ", "
                + peer3.cert[0][2]
                + ", "
                + peer3.cert[0][3]
                + "' !");

//Inicialização do Peer2
Peer peer2 = new Peer(ap1, as1, peer3);
peer2.ID = "Peer2";
peer2.Ch_Pb = "Kp2";
peer2.Acesso = "acesso sub-rede I";
peer2.ID_AP = "AP1";
peer2.cert[0][0] = "Kas2";
peer2.cert[0][1] = "Kp2";
peer2.cert[0][2] = "falso";
peer2.cert[0][3] = "acesso sub-rede A";
System.out.println(
       "O peer '"
                + peer2.ID
                + "' foi inicializado com a chave '"
                + peer2.Ch_Pb
                + "' e o certificado '"
                + peer2.cert[0][0]
                + ", "
                + peer2.cert[0][1]
                + ", "
                + peer2.cert[0][2]
                + ", "
                + peer2.cert[0][3]
                + "' !");

//Inicialização do Peer4
Peer peer4 = new Peer(ap2, as1, peer3);
peer4.ID = "Peer4";
peer4.Ch_Pb = "Kp4";
peer4.Acesso = "acesso sub-rede I";
peer4.ID_AP = "AP2";
peer4.cert[0][0] = "Kx";
peer4.cert[0][1] = "Kp4";
peer4.cert[0][2] = "falso";
peer4.cert[0][3] = "acesso sub-rede I";
System.out.println(
       "O peer '"
                + peer4.ID
                + "' foi inicializado com a chave '"
                + peer4.Ch_Pb
                + "' e o certificado '"
                + peer4.cert[0][0]
                + ", "
```

```
                                + peer4.cert[0][1]
                                + ", "
                                + peer4.cert[0][2]
                                + ", "
                                + peer4.cert[0][3]
                                + "' !");

                    //Inicia a execução das threads
                    peer4.start();
                    peer1.start();
                    peer3.start();
                    peer2.start();

            }
}
```

A.2. Resultado de Execução

O texto a seguir representa um resultado típico da execução do simulador construído.

```
Servidor de autenticação 'AS1' inicializado com a chave 'Kas1' !
Registro de autorização 1: 'Kas, acesso sub-rede I' criado !
Registro de autorização 2: 'Kas1, acesso sub-rede I' criado !
Registro de autorização 3: 'Kas2, acesso sub-rede A' criado !
Registro de autorização 4: 'Kas3, acesso sub-rede A' criado !
Registro de autorização 5: 'Kp1, acesso sub-rede I' criado !
AP 'AP1' inicializado !
AP 'AP2' inicializado !
O peer 'Peer1' foi inicializado com a chave 'Kp1' e o certificado 'Kas2, Kp1,
falso, acesso sub-rede A' !
O peer 'Peer3' foi inicializado com a chave 'Kp3' e o certificado 'Kas, Kp3,
verdadeiro, acesso sub-rede I' !
O peer 'Peer2' foi inicializado com a chave 'Kp2' e o certificado 'Kas2, Kp2,
falso, acesso sub-rede A' !
O peer 'Peer4' foi inicializado com a chave 'Kp4' e o certificado 'Kx, Kp4, falso,
acesso sub-rede I' !
O Peer4 foi associado com sucesso ao AP2 !
A identificação (chave pública) Kp4 do peer foi enviada pelo Peer4 para o AP2 !
A chave Kp4 do Peer4 foi registrada com sucesso no AP2 !
A identificação (chave pública) Kp4 do peer foi encaminhada do AP2 para o AS1 !
A mensagem 'TLS Start' destinada ao Peer4 foi enviada pelo AS1 para o AP2 !
A mensagem 'TLS Start' foi encaminhada pelo AP2 para o Peer4!
O Peer1 já estava associado ao AP1 !
A identificação (chave pública) Kp1 do peer foi enviada pelo Peer1 para o AP1 !
A chave Kp1 do Peer1 foi registrada com sucesso no AP1 !
A identificação (chave pública) Kp1 do peer foi encaminhada do AP1 para o AS1 !
A mensagem 'TLS Start' destinada ao Peer1 foi enviada pelo AS1 para o AP1 !
A mensagem 'TLS Start' foi encaminhada pelo AP1 para o Peer1!
O Peer3 foi associado com sucesso ao AP2 !
A identificação (chave pública) Kp3 do peer foi enviada pelo Peer3 para o AP2 !
A chave Kp3 do Peer3 foi registrada com sucesso no AP2 !
A identificação (chave pública) Kp3 do peer foi encaminhada do AP2 para o AS1 !
A mensagem 'TLS Start' destinada ao Peer3 foi enviada pelo AS1 para o AP2 !
A mensagem 'TLS Start' foi encaminhada pelo AP2 para o Peer3!
***Início do repouso da thread 'Peer4' !
***Início do repouso da thread 'Peer1' !
O Peer2 foi associado com sucesso ao AP1 !
A identificação (chave pública) Kp2 do peer foi enviada pelo Peer2 para o AP1 !
A chave Kp2 do Peer2 foi registrada com sucesso no AP1 !
A identificação (chave pública) Kp2 do peer foi encaminhada do AP1 para o AS1 !
A mensagem 'TLS Start' destinada ao Peer2 foi enviada pelo AS1 para o AP1 !
```

```
A mensagem 'TLS Start' foi encaminhada pelo AP1 para o Peer2!
***Início do repouso da thread 'Peer3' !
***Início do repouso da thread 'Peer2' !
*****Fim do repouso da thread 'Peer4' !
A requisição de acesso do tipo 'acesso sub-rede I' foi enviada pelo Peer4 para o
AP2 !
A requisição de acesso do tipo 'acesso sub-rede I' criada pelo Peer4 foi
encaminhada pelo AP2 para o AS1 !
A lista contendo as chaves autorizadas 'Kas Kas1 Kp1' destinada ao Peer4 foi
enviada pelo AS1 para o AP2 !
A lista contendo as chaves autorizadas 'Kas Kas1 Kp1' foi encaminhada pelo AP2 para
o Peer4 !
A cadeia de certificados 'null null null null' foi enviada pelo Peer4 para o AP2
juntamente com a requisição 'acesso sub-rede I' e a chave pública do peer 'Kp4'!
A cadeia de certificados 'null null null null' do Peer4 foi encaminhada pelo AP2
para o AS1 juntamente com a requisição 'acesso sub-rede I' e a chave pública do
peer 'Kp4'!
A mensagem 'TLS Alert' foi enviada pelo AS1 para o AP2, indicando que houve falha
no processo de autorização do Peer4 !
A mensagem 'TLS Alert' foi encaminhada pelo AP2 para o Peer4, indicando que o houve
falha no processo de autorização do Peer4 !
Um pacote vazio foi enviado pelo Peer4 para o AP2 em resposta ao 'TLS Alert'!
O pacote vazio do Peer4 foi encaminhado pelo AP2 para o AS1 em resposta ao 'TLS
Alert'!
A mensagem 'EAP-Failure' foi enviada pelo AS1 para o AP2, indicando que o processo
de autorização do Peer4 NÃO foi concluído com sucesso!
A mensagem 'EAP-Failure' foi encaminhada pelo AP2 para o Peer4, indicando que o
processo de autorização do Peer4 NÃO foi concluído com sucesso!
O Peer4 foi desassociado com sucesso do AP2 !
A requisição para emissão de certificado de autorização foi enviada pelo Peer4 para
o Peer3 !
*****Fim do repouso da thread 'Peer1' !
*****Fim do repouso da thread 'Peer3' !
A informação 'Emissão de certificado possível' foi enviada do Peer3 para o Peer4 !
A requisição de acesso do tipo 'acesso sub-rede I' foi enviada pelo Peer1 para o
AP1 !
A requisição para a emissão de certificado de autorização + chave foi enviada pelo
Peer4 para o Peer3 !
A requisição de acesso do tipo 'acesso sub-rede I' criada pelo Peer1 foi
encaminhada pelo AP1 para o AS1 !
Os certificados: 'Kas Kp3 verdadeiro acesso sub-rede I' e  'Kp3 Kp4 falso acesso
sub-rede I' foram enviados pelo Peer3 para o Peer4 !
A lista contendo as chaves autorizadas 'Kas Kas1 Kp1' destinada ao Peer1 foi
enviada pelo AS1 para o AP1 !
Os certificados: 'Kas Kp3 verdadeiro acesso sub-rede I' e  'Kp3 Kp4 falso acesso
sub-rede I' foram inseridos na matriz de certificados do Peer4 !
A lista contendo as chaves autorizadas 'Kas Kas1 Kp1' foi encaminhada pelo AP1 para
o Peer1 !
A cadeia de certificados 'null null null null' foi enviada pelo Peer1 para o AP1
juntamente com a requisição 'acesso sub-rede I' e a chave pública do peer 'Kp1'!
A cadeia de certificados 'null null null null' do Peer1 foi encaminhada pelo AP1
para o AS1 juntamente com a requisição 'acesso sub-rede I' e a chave pública do
peer 'Kp1'!
A mensagem 'EAP-Success' foi enviada pelo AS1 para o AP1, indicando que o processo
de autorização do Peer1 foi concluído com SUCESSO !
A mensagem 'EAP-Success' foi encaminhada pelo AP1 para o Peer1, indicando que o
processo de autorização do Peer1 foi concluído com SUCESSO !
A requisição de acesso do tipo 'acesso sub-rede I' foi enviada pelo Peer3 para o
AP2 !
A requisição de acesso do tipo 'acesso sub-rede I' criada pelo Peer3 foi
encaminhada pelo AP2 para o AS1 !
A lista contendo as chaves autorizadas 'Kas Kas1 Kp1' destinada ao Peer3 foi
enviada pelo AS1 para o AP2 !
A lista contendo as chaves autorizadas 'Kas Kas1 Kp1' foi encaminhada pelo AP2 para
o Peer3 !
```

A cadeia de certificados 'Kas Kp3 verdadeiro acesso sub-rede I' foi enviada pelo Peer3 para o AP2 juntamente com a requisição 'acesso sub-rede I' e a chave pública do peer 'Kp3'!
A cadeia de certificados 'Kas Kp3 verdadeiro acesso sub-rede I' do Peer3 foi encaminhada pelo AP2 para o AS1 juntamente com a requisição 'acesso sub-rede I' e a chave pública do peer 'Kp3'!
A mensagem 'EAP-Success' foi enviada pelo AS1 para o AP2, indicando que o processo de autorização do Peer3 foi concluído com SUCESSO !
A mensagem 'EAP-Success' foi encaminhada pelo AP2 para o Peer3, indicando que o processo de autorização do Peer3 foi concluído com SUCESSO !
*****Fim do repouso da thread 'Peer2' !
A requisição de acesso do tipo 'acesso sub-rede I' foi enviada pelo Peer2 para o AP1 !
A requisição de acesso do tipo 'acesso sub-rede I' criada pelo Peer2 foi encaminhada pelo AP1 para o AS1 !
A lista contendo as chaves autorizadas 'Kas Kas1 Kp1' destinada ao Peer2 foi enviada pelo AS1 para o AP1 !
A lista contendo as chaves autorizadas 'Kas Kas1 Kp1' foi encaminhada pelo AP1 para o Peer2 !
A cadeia de certificados 'null null null null' foi enviada pelo Peer2 para o AP1 juntamente com a requisição 'acesso sub-rede I' e a chave pública do peer 'Kp2'!
A cadeia de certificados 'null null null null' do Peer2 foi encaminhada pelo AP1 para o AS1 juntamente com a requisição 'acesso sub-rede I' e a chave pública do peer 'Kp2'!
A mensagem 'TLS Alert' foi enviada pelo AS1 para o AP1, indicando que houve falha no processo de autorização do Peer2 !
A mensagem 'TLS Alert' foi encaminhada pelo AP1 para o Peer2, indicando que o houve falha no processo de autorização do Peer2 !
Um pacote vazio foi enviado pelo Peer2 para o AP1 em resposta ao 'TLS Alert'!
O pacote vazio do Peer2 foi encaminhado pelo AP1 para o AS1 em resposta ao 'TLS Alert'!
A mensagem 'EAP-Failure' foi enviada pelo AS1 para o AP1, indicando que o processo de autorização do Peer2 NÃO foi concluído com sucesso!
A mensagem 'EAP-Failure' foi encaminhada pelo AP1 para o Peer2, indicando que o processo de autorização do Peer2 NÃO foi concluído com sucesso!
O Peer2 foi desassociado com sucesso do AP1 !
O usuário do Peer2 faz uma consulta pessoal ao usuário do Peer3 sobre a possibilidade de emitir um certificado com a autorização 'acesso sub-rede I' !
O usuário do Peer3 analisa o que foi solicitado, identifica que o peer pode atende-la e responde ao usuário do Peer2 !
O Peer3 solicita ao Peer2 a sua identificação (chave pública) e a requisição (lista de chaves autorizadas e tipo de acesso) necessários para a emissão do certificado !
A requisição para a emissão de certificado de autorização + chave foi enviada pelo Peer2 para o Peer3 !
Os certificados: 'Kas Kp3 verdadeiro acesso sub-rede I' e 'Kp3 Kp2 falso acesso sub-rede I' foram enviados pelo Peer3 para o Peer2 !
Os certificados: 'Kas Kp3 verdadeiro acesso sub-rede I' e 'Kp3 Kp2 falso acesso sub-rede I' foram inseridos na matriz de certificados do Peer2 !