Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Curso de Ciência da Computação

JACIARA SÁ SANTANA

Desenvolvimento de Aplicações Seguras na Plataforma JEE: o Modelo de Segurança para o Sistema Integrado de Apoio ao Paciente (SIAP)

São Luís
2014

JACIARA SÁ SANTANA

Desenvolvimento de Aplicações Seguras na Plataforma JEE: o Modelo de Segurança para o Sistema Integrado de Apoio ao Paciente (SIAP)

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para a obtenção do grau de BACHAREL em Ciência da Computação.

Orientador: Francisco José da Silva e Silva

Doutor em Ciência da Computação - UFMA

São Luís

2014

Santana, Jaciara Sá

Desenvolvimento de Aplicações Seguras na Plataforma JEE: o Modelo de Segurança para o Sistema Integrado de Apoio ao Paciente (SIAP) / Jaciara Sá Santana - 2014

92.p

1. Sistemas corporativos - mecanismos de segurança 2. Tecnologia Java EE. I.Título.

CDU 004.056

JACIARA SÁ SANTANA

Desenvolvimento de Aplicações Seguras na Plataforma JEE: o Modelo de Segurança para o Sistema Integrado de Apoio ao Paciente (SIAP)

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para a obtenção do grau de BACHAREL em Ciência da Computação.

Aprovado em 10 de setembro de 2014

BANCA EXAMINADORA

Francisco José da Silva e Silva

Doutor em Ciência da Computação - UFMA

Denivaldo Cicero Pavão Lopes

Doutor em Informática - UFMA

Samyr Beliche Vale

Doutor em Informática - UFMA

Aos meus pais, irmãos e à Silvia.

Resumo

Desenvolvedores de aplicativos distribuídos contam cada vez mais com uma variedade de poderosas APIs que reduzem a complexidade da atividade de desenvolvimento e melhoram a performance do software. Sistemas de negócio desenvolvidos sob a especificação Java EE tem a sua disposição um conjunto de serviços essenciais como gerenciamento de objetos distribuídos, transações e segurança, permitindo que desenvolvedores se concentrem mais na lógica de negócio da aplicação. Geralmente estes sistemas manipulam recursos sensíveis a segurança, por isso é importante protegê-los contra potenciais ameaças. Dentre os serviços de segurança oferecidos pela plataforma Java EE estão a comunicação segura, mecanismos de autenticação e autorização. Estes serviços podem ser configurados e implementados de várias maneiras, a autenticação, por exemplo, pode ser realizada de cinco formas. Nesta monografia descreve-se como os serviços disponíveis na plataforma Java EE foram utilizados para a implementação dos mecanismos de segurança projetados para o Sistema Integrado de Apoio ao Paciente (SIAP), um sistema de negócios voltado para a área da saúde. Este sistema foi desenvolvido com o objetivo principal de dar acesso a médicos sobre informações relativas a saúde cardíaca de pacientes localizados em regiões distantes. Através de uma central de atendimento instalada em uma instituição de saúde como o Hospital Universitário da Universidade Federal do Maranhão, médicos cardiologistas passam a ter acesso as informações coletadas (em particular o exame de ECG - EletroCardioGrama), ampliando o atendimento médico especializado até as cidades mais remotas e auxiliando as equipes locais em relação ao tratamento de saúde dos pacientes. Por manipular informações sobre a saúde de indivíduos, a central de atendimento requer a adição de mecanismos de proteção e segurança, sendo importante fonte para exemplificar e orientar desenvolvedores quanto a implementação da segurança em sistemas corporativos. As principais contribuições deste trabalho monográfico são, portanto, o desenvolvimento de um modelo de segurança para o SIAP e a descrição de como a tecnologia Java EE foi utilizada para a implementação dos mecanismos de segurança utilizados por este sistema.

Palavras-chaves: Sistemas corporativos. Mecanismos de segurança. Tecnologia Java EE.

Abstract

Developers of distributed applications rely increasingly on a variety of powerful APIs that reduce the complexity of developing and improving the performance of the software. Developed business systems under the Java EE specification has at its disposal a set of essential services such as managing distributed objects, transactions and security, allowing developers to focus more on the business logic of the application. Usually these systems handling sensitive security resources, so it is important to protect them against potential threats. Among the services security offered by the Java EE platform are secure communication, authentication and authorization mechanisms. These services can be configured and implemented in various ways, authentication, for example, can be performed in five forms. This monograph describes how the services available in the Java EE platform was used to implement the security mechanisms designed for the Sistema Integrado de Apoio ao Paciente (SIAP), a business system focused on health. This system was developed with the main objective of giving access to medical information about the cardiac health of patients located in remote regions. Through a call center located in an institution health as the University Hospital of the Federal University of Maranhão, cardiologists will have access to information collected (in particular the examination of ECG - ElectroCardioGram), expanding the specialized medical care to the most remote cities and assisting local teams in relation to the health care of patients. By manipulating information about the health of individuals, the call center requires the addition of mechanisms for protection and security, being an important source for exemplify and guide developers as implementing security in enterprise systems. The main contributions of this monograph are therefore the development of a security model for the SIAP and description of how Java EE technology was used to implement the security mechanisms used by this system.

Keywords: Business systems. Security mechanisms. Java EE technology.

Agradecimentos

Ao longo da vida, para alguns seres humanos, uma coisa intrigante parece acontecer: quanto mais tempo se vive, mais se desaprende a viver. Para estes seres a vida plena parece se encerrar quando se inicia a fase adulta, o tempo, que antes parecia infinito e se distribuia bem entre a família, os amigos, os estudos, as diversões e as reflexões, com o tempo se torna escasso.

Antes que saturem de suas vidas não vividas, Deus, sabiamente, coloca no caminho destes seres apáticos outros seres humanos incríveis para mostrar que toda essa "complexidade da vida moderna" não passa de um grande engodo, que a vida é um malabarismo permanente e que o resultado dessa mistura de dança, destreza, equilíbrio e graça vem de muito esforço e perseverança.

Parece contraditório comparar a vida com um espatáculo acrobático, no entanto, por mais complexo que possa parecer, o malabarismo resulta de um conjunto de movimentos simples que levam em consideração a sensibilidade de perceber que, se algo não vai bem, é só flexionar pro lado certo e o objeto que estava prestes a escapar se encaixa tão bem no movimento que ao final tudo sai como num perfeito ensaio. Apesar de ter levado tempo pra perceber que tudo é questão de simplicidade, enfim, inicia-se a partir de agora uma nova fase em minha vida: a fase da simplicidade que leva ao movimento permanente de viver. Agradeço as pessoas que, apesar de tão atribuladas, com a simples atitude de encaixar em suas vidas um pequeno espaço de tempo pra mim, me ensinaram, sem perceber, que na vida há tempo pra tudo, até para uma aluna que procrastina a conclusão de seu curso de graduação ou para uma irmã e amiga que não consegue seguir em frente e ser companheira na longa jornada de estudos para concursos públicos.

Neste espaço quero registrar minha gratidão a todos que fizeram e fazem parte desta conquista.

Em primeiro lugar, agradeço a DEUS, que sempre esteve presente em minha vida, guiou meus passos e iluminou meu caminho. Sinto-me envolvida por SUA proteção em todos os lugares por onde ando. Não fosse todo o conforto e alegria que traz ao meu

coração, a força para não desistir e superar os momentos difíceis, esta conquista jamais aconteceria.

A minha mãe, Maria Santana, que encarou o desafio de ser pai e mãe: cuidar de quatro filhos, trabalhar, administrar um lar e acolher parentes como se fossem seus filhos. Agradeço imensamente por todo amor, cuidado, compreensão e companheirismo dedicados a mim. Sua vida é exemplo de força, luta e perseverança para mim e para muitas mulheres.

Ao meu pai, Juracy. Apesar da distância física e emocional, sua história de vida é um exemplo de superação. Muito obrigada por todo seu apoio e confiança.

Aos meus irmãos: Jailson, Jandira e Jacira. Muito obrigada pelo companeheirismo, pela confiança e pelas responsabilidades que vocês tiveram que assumir em meu lugar para que eu tivesse todas as condições necessárias para estudar e chegar onde cheguei.

A Silvia. Obrigada por ter cuidado de mim e ter me acompanhado desde as minhas primeiras lições escolares, compartilhado comigo e com minha família os momentos bons e difíceis.

A grande amiga Regilaine, que é linda em todos os aspectos e nunca se esqueceu de mim, sempre me dando conselhos e força para superar os obstáculos.

Aos amigos da graduação. Gislaine, Paulo, Alan, Alex, Anilton, Jesseildo e Domingos, que agora parecem tão distantes, mas já estiveram muito próximos, compartilhando conhecimento, experiências e aflições.

Ao meu orientador, Francisco. Muito obrigada por nunca ter desistido de mim, pela sua paciência, pelos seus ensinamentos, por ser uma pessoa tão linda, inspiradora e admirada por todos que tem a oportunidade te conhecer.

Aos professores Denivaldo e Samyr por terem aceito fazer parte da banca da minha monografia.

Ao professor Carlos Eduardo Serra Portela. Agradeço pela paciência em esperar pela conclusão deste trabalho, me aconselhar e estimular, evitando que minha situação no curso se tornasse crítica.

A FINEP e ao CNPQ, financiadores do projeto no qual este trabalho está inserido.

Ao professor Allan Kardec Duailibe Barros, coordenador geral deste projeto.

A Alana e Wesley, que me fazem companhia no trajeto para o trabalho e me enriquecem com suas conversas inteligentes.

A Verônica, mulher polivalente. Um dia vou ler que você é grande destaque na produção cinematográfica nos principais jornais e revistas ao redor do mundo. Muito obrigada pelas mensagens de força e confiança.

Izabella e Gerlane, obrigada por tentarem me ajudar. Enfim, consegui terminar!

Àqueles cujo nome não foi listado aqui, peço desculpa por não tê-los trazido a memória neste exato momento, mas saibam que estão no meu coração e que ele sempre se alegrará toda vez que eu pensar em vocês.

Como posso retribuir ao Senhor toda a sua bondade para comigo?

Salmos, capítulo 116, versículo 12.

Sumário

1 Introdução

Uma das principais vantagens oferecidas por sistemas distribuídos é o acesso e compartilhamento de recursos remotos de forma controlada e eficiente. Exemplos clássicos de recursos compartilhados são dados, páginas WEB, impressoras, etc. Este tipo de sistema é formado por diversos computadores independentes conectados através de uma rede que se apresentam como um sistema único e coerente. Um dos principais desafios relacionados a aplicativos distribuídos diz respeito a segurança, uma vez que é necessário aplicá-la a todos os componentes que fazem parte do aplicativo de modo consistente, uma falha de segurança em apenas um componente pode inviabilizar todo o sistema [1]. Adicionalmente, deve-se somar a esta circunstância o fato de sistemas distribuídos usualmente manipularem recursos sensíveis que podem ser acessados por vários usuários ou atravessar redes abertas ou desprotegidas [3], estes aspectos evidenciam o quanto pode ser complexa a implantação de medidas segurança em sistemas distribuídos e como estas medidas são essenciais para torná-los viáveis. Medidas de segurança geralmente são aplicadas através da adição de canais seguros de comunicação, controle de acesso e gerenciamento de segurança, conceitos explicados sucintamente nos próximos três parágrafos.

Canais seguros garantem a comunicação segura entre usuários ou processos que se encontram em locais diferentes e protegem o sistema distribuído contra ameaças de interceptação, modificação e invenção e interrupção. Para alcançar estes objetivos é necessário que o canal seguro execute mecanismos de autenticação e garanta confidencialidade e integridade de mensagens [4], assegurando que as mensagens transmitidas através dele sejam acessadas apenas por entidades legítimas cuja identidade já foi comprovada e não sofram nenhuma alteração em trânsito, saindo da origem e chegando ao destino intactas. A autenticação é o mecanismo usado para verificar a identidade declarada por uma entidade que tenta acessar recursos sensíveis do sistema e é imprescindível a comunicação segura, uma vez que as entidades que tentam trocar mensagens de forma confiável precisam se reconhecer mutuamente antes de formar o canal que manterá as mensagens íntegras e confidenciais [5]. Integridade diz respeito a permitir apenas a entidades autorizadas o direito de alterar o conteúdo das mensagens transmitidas, assegurando que partam da origem e cheguem ao destino intactas, já confidencialidade se refere a assegurar

que apenas estas entidades terão acesso ao conteúdo das mensagens.

Controle de acesso consiste em verificar se o usuário ou processo tem direitos de acesso a determinado recurso, se o acesso for permitido o direito é concedido através de autorização e o usuário ou processo que fez a requisição poderá executar as operações solicitadas sobre o recurso protegido. Para verificar direitos de acesso é preciso estabelecer previamente uma política de segurança onde as regras de direito de acesso estejam bem definidas, esta política descreve quais ações podem ser executadas e quais devem ser proibidas para cada uma das entidades que acessam o sistema, assim que a política é estabelecida é necessário determinar os mecanismos de segurança que serão usados para impô-la e a forma como estes serão usados. Os principais mecanismos de segurança são: criptografia, autenticação, autorização e auditoria [1]. Criptografia é uma técnica utilizada para disfarçar dados, transformando-os em algo ilegível para o atacante. Autenticação, como explicado anteriormente, é o meio pelo qual entidades que se comunicam, tais como cliente e servidor, usam para provar suas identidades mutuamente [6]. Autorização é a concessão de direitos de acesso a um recurso para um conjunto limitado de usuários ou processos. Auditoria é o armazenamento inviolável de dados sobre eventos relacionados a segurança com o intuito de detectar a efetividade da política e dos mecanismos de segurança adotados [3]. Existem vários modelos de controle de acesso, no entanto, pode-se considerar que todos eles convergem para um modelo genérico que consiste nos componentes sujeito, objeto e monitor de referência, mostrado na figura 1.1. O sujeito é a entidade que faz requisições, o objeto é o recurso alvo das requisições e o monitor de referência é o componente que armazena informações de direitos de acesso, possui um mapeamento que registra as operações que cada sujeito pode executar, ou seja, armazena as regras de direito de acesso.

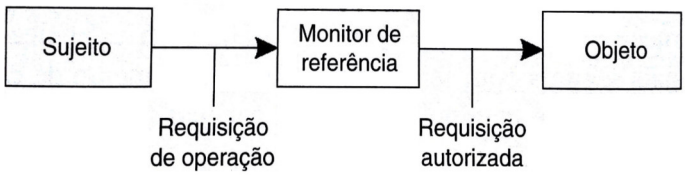

Figura 1.1: modelo geral de controle de acesso a objetos [1].

Gerenciamento de segurança é um processo que está relacionado a administrar a adição ou remoção de usuários e grupos de usuários do sistema, administrar a distribuição de chaves criptográficas e gerenciar direitos de acesso [1]. O gerenciamento de

chaves públicas é um típico exemplo de gerenciamento de chaves criptográficas, estas chaves podem ser distribuídas através de certificados de chave pública que associam a chave a identidade de seu proprietário. Estes certificados são assinados por autoridades de certificação confiáveis que se organizam hierarquicamente de forma a facilitar a distribuição e verificação da validade dos certificados.

Os sistemas distribuídos baseados na WEB são o tipo de aplicativo de redes mais conhecidos entre usuários finais. Mais do que documentos em formato de páginas, sistemas WEB também podem oferecer conteúdo dinâmico e serviços [1]. Para suportar funcionalidades cada vez mais complexas, muitos sistemas baseados na WEB estão sendo projetados seguindo modelos de arquitetura cada vez mais sofisticadas e baseadas em objetos distribuídos, que frequentemente precisam ser protegidos através de canais seguros de comunicação, controle de acesso e gerenciamento de segurança. Um bom exemplo deste modelo são os softwares corporativos desenvolvidos para operar sobre a plataforma Java Enterprise Edition (Java EE), atualmente na versão 7.

A plataforma Java EE define as especificações necessárias para a construção de servidores de aplicação projetados para suportar aplicativos corporativos formados por componentes distribuídos, tais como servlets, páginas JavaServer Pages (JSP) ou JavaServer Faces (JSF) e Enterprise Java Beans (EJB)[2]. Em relação ao último componente citado é importante destacar sua natureza distribuída: EJBs são objetos de negócio que recebem requisições provenientes de entidades (aplicativos externos, EJBs ou outros componentes), executam operações sobre recursos compartilhados e, caso seja necessário, se comunicam com outros beans para auxiliar na execução de suas tarefas, colaborando com o funcionamento do sistema como um todo.

As especificações da plataforma Java EE definem uma série de protocolos e extensões necessárias a construção de servidores de aplicação que funcionem como um ambiente de execução para sistemas de negócios, estes servidores devem implementar uma variedade de serviços de gerenciamento, chamados de serviços primários, que visam facilitar as atividades dos desenvolvedores de sistemas corporativos para que os mesmos concentrem seus esforços apenas na lógica de negócios deste tipo de sistema. Desta forma, pode-se inferir que a plataforma Java EE define uma camada de software intermediária (middleware) voltada para a execução de sistemas distribuídos que seguem suas especificações e oferece um conjunto de facilidades que a transformam numa camada homogênea que minimiza a preocupação do desenvolvedor com aspectos que estão fora do contexto

do negócio [2]. São exemplos de serviços primários o gerenciamento de concorrência, transações, mensagens assíncronas, segurança, etc.

Dadas as vantagens de se desenvolver aplicações corporativas na especificação Java EE, o Laboratório de Sistemas Distribuídos da Universidade Federal do Maranhão optou por esta solução no desenvolvimento do Sistema Integrado de Apoio ao Paciente (SIAP), um sistema voltado para a área da saúde, com foco no acesso a informações sobre a saúde cardíaca de pacientes de regiões carentes de atendimento especializado em saúde. O SIAP foi projetado para utilizar uma base de dados que pode ser alimentada com informações provenientes de equipamentos e dispositivos móveis (aparelho eletrocardiográfico, ou ECG, e smartphone), registradas por membros das equipes de atendimento em saúde do local. Com as informações relacionadas ao desempenho cardíaco de pacientes enviadas via dispositivo móvel a base de dados do sistema, cardiologistas do Hospital Universitário da Universidade Federal do Maranhão podem realizar operações como a emissão de laudos dos ECGs e registro de anamnese, por exemplo, ajudando as equipes locais em relação a procedimentos de auxílio a diagnose e terapêutica. A figura 1.2 dá uma visão geral sobre o funcionamento do sistema e a interação entre suas diferentes partes.

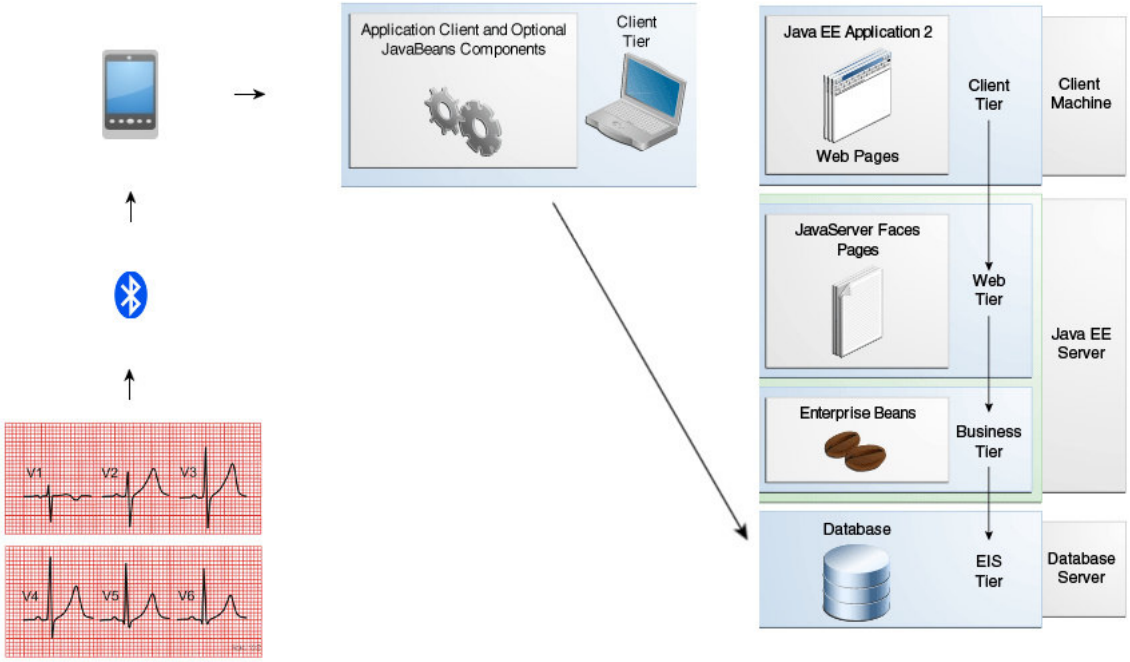

Figura 1.2: visão geral do funcionamento do sistema.

O foco deste trabalho está no sistema Web que integra o SIAP, o qual per-

mite gerar prontuários eletrônicos de pacientes, visando facilitar o registro e o acesso às informações, sinais e imagens relacionados à saúde e à assistência. Uma vez que as informações manipuladas através do SIAP exigem cuidados especiais em relação a segurança, serviços primários de autenticação, autorização e comunicação segura da plataforma, além da implementação de mecanismo de auditoria e assinatura digital foram necessários para viabilizar o uso do sistema. A ênfase deste trabalho está no desenvolvimento de um modelo de segurança para o SIAP e a descrição de como a tecnologia Java EE foi utilizada para a implementação dos mecanismos de segurança utilizados por este sistema.

1.1 Objetivos

Este trabalho monográfico tem por objetivo geral o desenvolvimento de um modelo de segurança, bem como a implementação dos mecanismos de segurança necessários para a manutenção da segurança das informações mantidas pelo software do projeto SIAP.

Os objetivos específicos desta monografia são:

1. Estudo dos serviços de segurança disponíveis na plataforma Java EE;
2. Definição de um modelo de segurança para o sistema SIAP;
3. Definição e implementação dos mecanismos de segurança requeridos pelo sistema SIAP;
4. Descrever como os serviços de segurança da plataforma Java EE foram utilizados para implementar os mecanismos de segurança a serem definidos para o SIAP.

1.2 Estrutura da Monografia

O presente trabalho irá tratar da implementação de mecanismos de segurança em sistemas de negócio e está organizado da seguinte forma:

Capítulo 2: Fundamentação teórica. Apresenta aspectos de segurança, criptografia, autenticação e gerenciamento de segurança;

Capítulo 3: Tecnologias utilizadas. Descreve a plataforma Java EE e seus componentes, as ferramentas e APIs necessárias para adicionar segurança a um aplicativo Java EE;

Capítulo 4: Modelo e implementação das primitivas de segurança do SIAP. Apresenta a solução de segurança modelada para o sistema e o modo como os mecanismos de segurança foram implementados para obedecer ao modelo proposto;

Capítulo 5: Conclusões.

2 Fundamentação Teórica

Atualmente muitos aplicativos permitem acesso a recursos remotos que precisam ser protegidos contra o acesso não autorizado, este capítulo irá abordar conceitos de segurança, criptografia, autenticação e gerenciamento de segurança a fim de formar a base teórica necessária a compreensão da arquitetura de segurança por trás de aplicativos de negócio Java EE, bem como a solução de segurança adotada pelo sistema SIAP.

2.1 Segurança

Para um sistema computacional ser considerado seguro ele deve ser confiável, o que significa assegurar [7]:

1. Confidencialidade: apenas as partes autorizadas tem acesso as informações do sistema;
2. Integridade: as alterações são realizadas apenas por quem tem autorização, assim aquilo que é transmitido chega ao destino sem alterações;
3. Legitimidade: os recursos do sistema computacional são usados apenas por quem tem direito de acesso e da forma prevista de uso para cada entidade autorizada;
4. Disponibilidade: o sistema garante a entrega de seus serviços a usuários autorizados sem degradação de acesso.

Deve-se unir as propriedades nomeadas acima o fato de um sistema computacional seguro também ser capaz de proteger seus recursos contra todas as possíveis ameaças a segurança e ter uma política bem definida[1]. Uma ameaça é uma potencial violação decorrente de uma vulnerabilidade não tratada[5], existem quatro tipos de ameaça a segurança [1]:

1. Modificação, consiste na alteração não autorizada de dados ou serviços para que se comportem de um modo para o qual não foram projetados;

2. Interceptação, situação onde um terceiro acessa dados ou serviços sem autorização das partes legítimas;

3. Interrupção, ocorre quando o invasor torna os dados ou serviços indisponíveis, inutilizáveis ou os destroem propositalmente;

4. Invenção, resume-se a adição de dados que jamais existiriam a uma mensagem original.

Sistemas computacionais podem ser protegidos contra ameaças através do uso de canais seguros de comunicação, controle de acesso e gerenciamento de segurança. Tudo isso pode ser obtido através de uma política de segurança bem definida e da adição dos mecanismos adequados para impô-la. Os principais mecanismos de segurança são: criptografia, autenticação, autorização e auditoria, assuntos explicados resumidamente no capítulo anterior e estendidos no presente capítulo.

2.2 Criptografia

Criptografia é uma técnica utilizada para cifrar mensagens que as transformam em algo ilegível para o atacante. Existem vários algoritmos de criptografia, os principais se fundamentam em métodos parametrizados por chaves criptográficas e podem ser divididos em dois grupos: algoritmos de chaves simétricas e algoritmos de chaves assimétricas, também conhecidos como de chave pública[6]. Os simétricos usam a mesma chave para cifrar e decifrar mensagens, esta chave é compartilhada apenas entre os componentes que se comunicam secretamente e deve ser mantida em sigilo, já os algoritmos assimétricos usam chaves distintas para encriptar e decriptar as mensagens, uma das chaves deve ser mantida em segredo enquanto sua complementar pode ser divulgada publicamente.

Data Encription Standard (DES) e Rinjdael são exemplos de algoritmos simétricos [6]. Diffie-Hellman, RSA (Rivest, Shamir e Adleman) e DSS (Digital Signature Standard) são exemplos de algoritmos de criptografia assimétrica, estes são baseados em propriedades matemáticas que tornam computacionalmente inviável descobrir o valor de uma das chaves conhecendo-se apenas o valor de sua chave complementar. Além desta, outras propriedades devem ser observadas nos algoritmos assimétricos, para explicá-las melhor a seguinte notação será usada: K_A^+ e K_A^- são as chaves pública e privada do usuário A,

respectivamente, E é o algoritmo de criptografia, D o algoritmo de descriptografia e C a mensagem criptografada através do algoritmo E usando uma das chaves (K_A^+ ou K_A^-). As propriedades obrigatórias à criptografia assimétrica são [8]:

1. Deve ser computacionalmente fácil para um usuário A gerar um par de chaves pública e privada (K_A^+ e K_A^-);

2. Deve ser computacionalmente fácil para um usuário B, conhecendo a chave pública de A e a mensagem m, produzir o texto criptografado $C = E_{K_A^+}(m)$;

3. Deve ser computacionalmente fácil para o receptor A descriptografar com sua chave privada, K_A^-, o texto criptografado com sua chave pública, K_A^+, e recuperar a mensagem original $m = D_{K_A^-}(C) = D_{K_A^-}[E_{K_A^+}(m)]$;

4. Deve ser computacionalmente inviável para um oponente, conhecendo apenas a chave pública K_A^+ e o texto criptografado C ($C = E_{K_A^+}(m)$), recuperar o texto original m;

5. Os algoritmos de criptografia e descriptografia podem ser aplicados em qualquer ordem, ou seja: $m = E_{K_A^+}[D_{K_A^-}(m)] = D_{K_A^-}[E_{K_A^+}(m)]$.

De um modo geral, os algoritmos criptográficos devem ser projetados de forma a tornar computacionalmente inviável deduzir o valor da chave utilizada quando se tem em mãos apenas a mensagem cifrada e a mensagem em texto claro que a originou, ou seja: sendo E o algoritmo criptográfico, P a mensagem em texto claro, K a chave criptográfica (simétrica, pública ou privada) e $E_K(P)$ a mensagem P criptografada com K, deve ser computacionalmente impossível encontrar o valor de K quando se conhece apenas P e $E_K(P)$. Outro requisito importante a estes algoritmos é a resistência a colisão, que pode ser explicada da seguinte forma: quando se tem apenas o algoritmo E e uma mensagem P é impossível encontrar, por computação, duas chaves distintas (K e K'), que produzam a mesma mensagem cifrada, ou seja, ter $E_K(P) = E_{K'}(P)$[1].

Além da classificação de algoritmos de criptografia em simétricos e assimétricos, há também outra classificação, ainda mais abrangente, que os subdivide em dois grupos: algoritmos de criptografia reversível e algoritmos de criptografia irreversível [5]. No primeiro grupo se encontram os simétricos e assimétricos, no segundo estão os algoritmos de resumo de mensagem, também conhecidos como funções hash. Os algoritmos de criptografia irreversível cifram as mensagens de um modo que é impossível recuperar a mensagem

original a partir do resumo obtido. A integridade de mensagens é uma importante propriedade obtida através das funções hash, uma pequena alteração na mensagem dada como entrada para a função produz um valor de saída completamente diferente daquele gerado com a mensagem original. Resumidamente, podemos enumerar as seguintes características como fundamentais as funções hash [6]:

1. Dada uma função hash H e uma mensagem m, é muito fácil obter h = H(m);
2. Impossível haver m e m', com $m \neq m'$, tal que H(m') = H(m)
3. Se h = H(m) for fornecido, será impossível encontrar m;
4. Uma função hash H recebe uma mensagem de tamanho arbitrário m como entrada e produz uma cadeia de bits h de tamanho fixo: h = H(m).

Com a criptografia é possível obter confidencialidade, integridade de dados, proteger sistemas contra ameaças de interceptação, modificação e invenção, estabelecer troca de chave de sessão, etc. Estas características evidenciam a criptografia como mecanismo fundamental para a comunicação segura.

2.3 Gerenciamento de Chaves

O gerenciamento de chaves faz parte do gerenciamento de segurança de um sistema computacional e, particularmente, trata de questões sobre o estabelecimento e distribuição de chaves criptográficas de modo seguro e formas de verificar a autenticidade e validade de uma chave.

2.3.1 Gerenciamento de Chave Simétrica

Na criptografia com chave simétrica o gerenciamento de chaves compreende garantir um ambiente onde a distribuição de chaves ocorre de modo confidencial, mediante autenticação. Assim, para funcionar corretamente, o gerenciamento de chaves deve garantir às entidades envolvidas, um indivíduo ou um Centro de Distribuição de Chaves (Key Distribution Center - KDC), por exemplo, que suas chaves secretas foram compartilhadas, sigilosamente, apenas com outras entidades que também passaram por autenticação e que tem permissão para usá-las [1].

Em relação a chave secreta compartilhada, recomenda-se que seu uso não seja exposto por um longo tempo, esta exposição deve ser minimizada pela substituição da chave secreta por uma chave de sessão tão logo a autenticação tenha sido concluída ou durante o processo de autenticação. Chaves de sessão são usadas apenas uma vez e por um período de tempo determinado, no máximo o tempo em que a conexão permanece ativa.

A criptografia de chave simétrica com Centro de Distribuição de Chaves (KDC) tem o KDC como um intermediário de confiança responsável por promover a autenticação, criar chaves de sessão, distribuí-las de forma segura e administrar chaves secretas compartilhadas, conforme veremos na subseção 2.4.2 deste capítulo. Todos que estabelecem comunicação segura via KDC confiam nesta entidade e registram suas chaves secretas compartilhadas nela. Um esquema alternativo de estabelecimento de chave compartilhada amplamente utilizado é o algoritmo conhecido como **Troca de Chaves Diffie-Hellman** [6]. A figura 2.1 representa o processo, que pode ser explicado, resumidamente, desta forma:

1. Alice e Bob acordam sobre dois números grandes, n e g, que estão relacionados por propriedades matemáticas. Os números n e g podem ser divulgados publicamente.

2. Alice escolhe um número grande, x, enquanto Bob escolhe outro número grande, y. Cada um guarda seu número em segredo.

3. Alice envia para Bob n, g e g^xmod n, através da mensagem 1, enquanto Bob responde com g^ymod n, através da mensagem 2, ambas podem ser transmitidas em texto aberto.

4. Cada um agora pode calcular g^{xy}mod n.

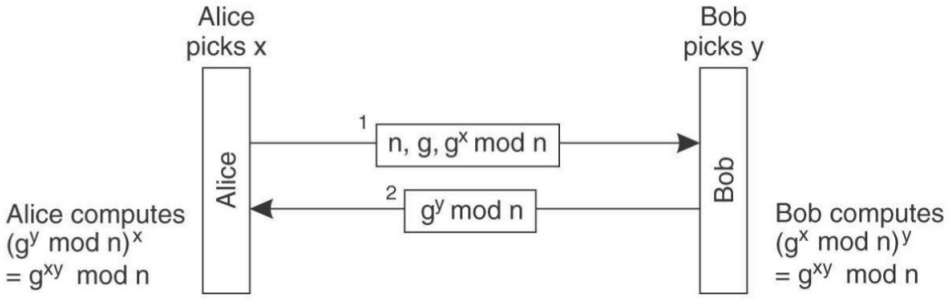

Figura 2.1: troca de chaves Diffie-Hellman [1].

No protocolo Diffie-Hellman os valores de g^xmod n e g^ymod n funcionam como chaves públicas, x e y como chaves privadas e g^{xy}mod n como chave de sessão, compartilhada somente entre Alice e Bob, que passará a ser utilizada para criptografar todas as mensagens trocadas entre estes dois personagens enquanto a sessão permanecer ativa.

2.3.2 Gerenciamento de Chave Pública

O gerenciamento de chaves na criptografia de chave pública preocupa-se com formas de verificar com facilidade a autenticidade e a validade de chaves públicas e prover estruturas que viabilizem o acesso a elas. Se houver violação de qualquer chave privada correspondente a uma chave pública gerenciada, o sistema de gerenciamento deve apresentar mecanismos para desativar e divulgar informações sobre chaves publicas que foram invalidadas. Não há confidencialidade na distribuição de uma chave pública, apenas a autenticação é necessária, ou seja, é preciso certificar-se de que ela realmente pertence a quem se apresenta como seu proprietário. O acesso a uma chave privada deve ser confidencial e passar por autenticação. Usualmente a distribuição de chave pública ocorre por certificados de chave pública [5].

Certificados de chave pública vinculam uma chave pública a identidade de uma entidade particular e contribuem para viabilizar o uso da criptografia de chaves assimétricas. O vínculo entre uma chave pública e uma identidade é comumente realizado por Autoridades Certificadoras (Certification Authority - CA), que devem dispor de mecanismos para verificar se as informações apresentadas por uma entidade realmente comprovam sua identidade antes de emitir um certificado para ela. Além de emitir certificados, uma CA também deve ser capaz de revogá-los e ter uma infraestrutura que permita fácil acesso àqueles que foram emitidos e a lista dos que foram revogados. Cada certificado possui a chave pública de seu proprietário e informações suficientes para identificar de maneira única seu dono, estas informações, junto a outras que identificam a autoridade que o emitiu, são assinadas pela própria CA. A figura 2.2 representa o processo de emissão de um certificado por uma autoridade [6]. A assinatura da CA sobre a chave pública e os dados de identidade citados é anexada ao próprio certificado, permitindo averiguar a qualquer tempo a validade do mesmo, basta ter acesso a chave pública da CA que o emitiu, também disponibilizada através de certificado.

O modelo de confiança hierárquico é o mais difundido quando se trata da or-

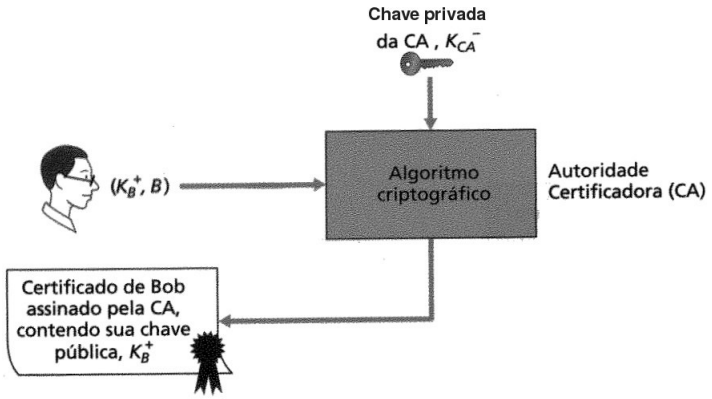

Figura 2.2: emissão de certificado mediado por uma Autoridade Certificadora [4].

ganização de Autoridades Certificadoras, nele cada CA possui um certificado de chave pública que é assinado por outra CA considerada ainda mais confiável. A cada nível hierárquico acima o grau de confiança aumenta, até o ponto em que se chega às autoridades com grau máximo de confiança, cujos certificados são auto-assinados, estas autoridades são conhecidas também como âncoras de confiança. Muitos certificados de âncoras e de autoridades intermediárias amplamente conhecidas já vem instalados em programas navegadores da web, estes certificados são de CA's consideradas confiáveis pelos fornecedores de navegador, no entanto a decisão final sobre a confiança cabe a quem irá fazer uso da chave pública. Outra forma de obter certificados de autoridades é através de locais públicos conhecidos e confiáveis, como servidores mantidos pela própria CA. Geralmente CA's são administradas por instituições consideradas idôneas, a confiança depositada numa CA vai depender da confiança que as pessoas tem na instituição [4].

O processo de validação de um certificado digital passa pela fase de verificação de integridade, etapa em que é imprescindível o uso da chave pública da autoridade que o assinou, a qual também pode ser obtida através de certificado digital. Uma vez obtida a chave pública da autoridade, o passo seguinte é descriptografar a assinatura presente no certificado em análise de autenticidade, o resultado deve ser o hash de todas as informações presentes no certificado, exceto a própria assinatura. Após obter o hash, o próximo passo é executar localmente uma função hash sobre as informações presentes no certificado, exceto a assinatura, e comparar o resultado local com o hash extraído da assinatura na fase anterior. Se os resultados forem iguais, então a autenticidade e a integridade do certificado está parcialmente comprovada. A indicação sobre qual função de resumo de mensagem deve ser utilizada está presente no próprio certificado. Se não

houver confiança suficiente na autoridade que assinou o certificado final, então o mesmo processo de validação deve ser aplicado recursivamente ao longo da cadeia de certificados da hierarquia, até que se chegue ao topo, caso seja necessário. O caminho percorrido durante este processo é conhecido como caminho de certificação ou cadeia de confiança. Averiguar a validade de um certificado envolve também consultar a lista de certificados revogados (LCR) da autoridade que o emitiu para se certificar de que as chaves privadas usadas nas assinaturas ao longo da cadeia permanecem intactas. Além disso, informações que podem estar presentes no certificado, como período de validade e atributos como idade, número de CPF, por exemplo, podem ser exigidos durante a validação.

A verificação de integridade descrita acima pode ser encontrada no RFC 1422, um documento que apresenta, em linhas gerais, padrões, procedimentos, convenções e recomendações que devem ser suportados por qualquer arquitetura e infraestrutura de gerenciamento de chave pública baseada em certificados. Os procedimentos e convenções propostos são bastante generalistas, capazes de viabilizar uma infraestrutura com potencial para funcionar em conjunto com qualquer protocolo de transmissão de dados pela rede. O referido documento propõe que toda arquitetura deve estar fundamentada numa política de certificação ampla, clara e bem articulada. Procedimentos sobre como validar certificados e como usá-los para prover integridade e autenticação de mensagem, como emitir e distribuir certificados e listas de certificados revogados, convenções para a hierarquia de certificação, as entidades envolvidas na arquitetura e seus respectivos papeis e responsabilidades, são exemplos do que é descrito no rfc 1422 [9]. A arquitetura de gerenciamento de chaves descrita no RFC 1422 é compatível com o framework de autenticação descrito na especificação X.509.

2.4 Autenticação

Autenticação é um processo através do qual se verifica se a identidade apresentada por uma entidade comprova que ela é realmente quem diz ser. Realizar este processo remotamente é mais complexo do que efetua-lo presencialmente, uma vez que os próprios meios que transmitem os dados necessários a autenticação podem ser alvo de ataques que tentem alterar ou extraviar os dados [6]. Desta forma, assegurar integridade de dados é um processo indispensável durante e após a autenticação, sem integridade é inútil saber a origem de uma mensagem se não há como garantir que ela não foi alterada

durante a transmissão. De modo semelhante, também é inútil ter garantia de que uma mensagem foi transmitida sem alterações se não há como comprovar sua origem. Vários protocolos de autenticação foram criados para solucionar problemas desta natureza, nesta seção veremos os três principais protocolos de autenticação.

2.4.1 Autenticação baseada em chave secreta compartilhada

Protocolos de autenticação baseados em chave secreta compartilhada partem da premissa de que as entidades que buscam se autenticar mutuamente já possuem uma chave secreta comum que foi distribuída previamente de forma segura. Além desta, outra abordagem comum a autenticação através de chave secreta compartilhada é o protocolo **desafio-resposta**, onde um dos lados envia um desafio ao outro, as respostas aos desafios só estarão corretas se cada um realmente tiver posse da chave compartilhada. Para explicar este protocolo de autenticação a seguinte notação será usada: A e B são as identidades das entidades Alice e Bob, respectivamente, $K_{A,B}$ é a chave compartilhada e R_A e R_B os desafios enviados por cada um.

As mensagens trocadas durante a execução do protocolo são apresentadas na figura 2.3. Na primeira mensagem Alice envia sua identidade A para Bob, indicando que deseja estabelecer um canal de comunicação. Para ter certeza que se trata de Alice, Bob envia-lhe um desafio, R_B, por meio da mensagem 2. Um desafio consiste de uma mensagem cujo principal objetivo é a autenticação: só poderá enviar a resposta correta (o desafio criptografado) aquele que realmente tiver posse da chave secreta, a qual é preservada em sigilo pelos pares que a possuem.

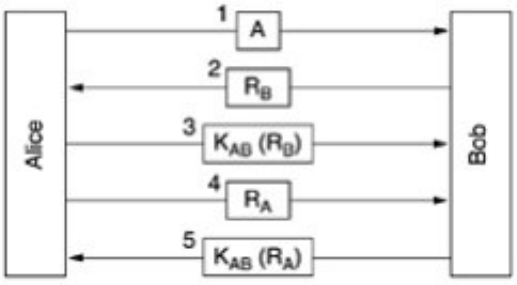

Figura 2.3: autenticação baseada em chave secreta compartilhada [6].

Ainda na figura 2.3, a mensagem 3 é a resposta a mensagem anterior: o desafio R_B criptografado com a chave compartilhada $K_{A,B}$. Ao receber a mensagem 3 Bob a

descriptografa e verifica se seu conteúdo corresponde ao desafio R_B enviado anteriormente. Se a resposta estiver correta, então Bob estará convencido de que Alice é o interlocutor. As mensagem 4 e 5 servem para que Alice verifique se está realmente falando com Bob, assim, de forma semelhante a Bob, Alice envia um desafio, R_A, e recebe uma resposta, $K_{A,B}(R_A)$. Caso a resposta de Bob esteja correta, então as duas partes estarão autenticadas. Após a autenticação mútua, Alice ou Bob poderá gerar uma chave de sessão, criptografa-la com a chave compartilhada $K_{A,B}$ e enviá-la, de modo que os dois passem a se comunicar temporariamente apenas com a nova chave de sessão.

Frequentemente a mensagem que compõe o desafio é um número extenso e aleatório, que é usado apenas uma vez. Nestas condições, o desafio é chamado de nonce e, além de funcionar como etapa da autenticação, também serve como método de prevenção contra ataques por reflexão. No ataque por reflexão o impostor guarda mensagens transmitidas durante o processo de autenticação de uma entidade legítima e as reproduz posteriormente a fim de conseguir obter o acesso. Como através do uso de nonces há um controle sobre os números aleatórios trocados, cada um dos pares do processo saberá identificar quando está recebendo uma mensagem repetida [6].

Na figura 2.4 o atacante, Trudy, intercepta a mensagem de apresentação de Alice e inicia uma segunda sessão se passando por Bob, esta segunda sessão será usada para induzir Alice responder aos seus próprios desafios, fazendo com que Trudy devolva as respostas corretas. Ao final do processo, Trudy tem duas sessões autenticadas com Alice.

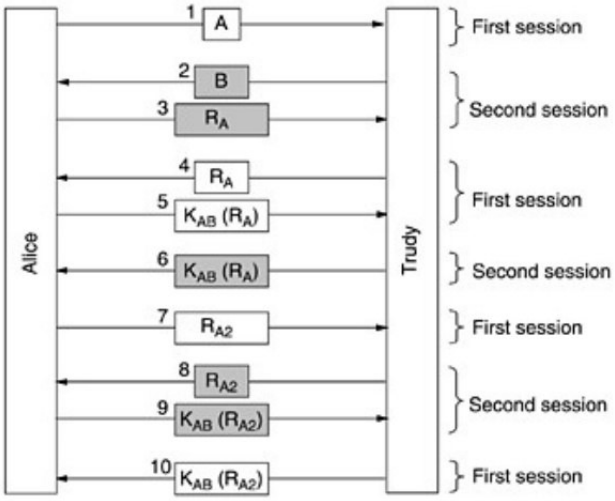

Figura 2.4: ataque por reflexão [6].

Um protocolo alternativo, imune a este tipo de ataque, é apresentado na figura 2.5, nele as respostas aos desafios são estruturas de dados HMAC. Estruturas HMAC são formadas pela concatenação dos desafios de cada uma das partes com a chave compartilhada, o resultado da concatenação é criptografado com uma função hash que protege as informações secretas transmitidas. A eficiência deste protocolo se deve a impossibilidade de cada um dos lados induzir o outro a responder com um valor de sua escolha, além disso, há pelo menos um valor secreto na estrutura, a chave compartilhada $K_{A,B}$. Com os valores expostos (R_A e R_B) e o valor secreto, cada lado será capaz de reproduzir o hash correspondente e comparar com a resposta enviada.

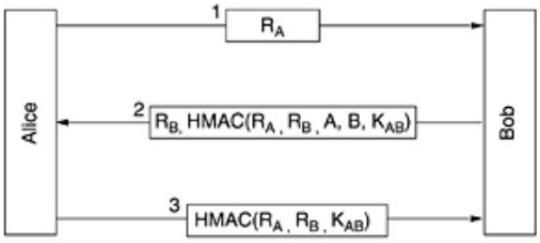

Figura 2.5: autenticação com HMAC [6].

2.4.2 Autenticação via centro de distribuição de chaves

Neste método de autenticação cada uma das entidades que pretende estabelecer comunicação compartilha uma chave criptográfica com um componente chamado de centro de distribuição de chaves (Key Distribution Center - KDC), esta chave secreta é conhecida apenas pelo KDC e pela entidade que deseja dar início a comunicação segura e está relacionada de maneira única com o seu proprietário. O KDC guarda uma relação de identidade entre cada chave e seu proprietário, esta relação, aliada a idoneidade atribuída ao KDC, formam a base fundamental de confiança deste processo de autenticação.

O compartilhamento de chave com o KDC minimiza problemas de escalabilidade, pois cada nó precisará gerenciar apenas a chave que compartilha com o KDC, ao contrário da autenticação baseada em chave secreta compartilhada, onde cada nó precisará manter as chaves secretas que compartilha com os outros $N - 1$ nós do sistema, havendo no total $\frac{N(N-1)}{2}$ chaves no sistema. Na autenticação via centro de distribuição de chaves, há apenas N chaves em todo o sistema e a carga de gerenciamento fica sobre o KDC [7].

A figura 2.6 representa o protocolo de autenticação Otway-Rees simplificado, nele A e B são as identidades das partes que desejam se comunicar, K_A e K_B são suas chaves secretas compartilhadas. R_A e R_B são os desafios que uma parte lança a outra para comprovar que a tentativa de estabelecer o canal seguro de comunicação está acontecendo em tempo real e que não se trata de uma repetição. R é um número aleatório, um nonce, que tem a função de ser o identificador comum as mensagens 1 e 2, servindo-se ao papel de vincular uma mensagem a outra e provar ao KDC que elas foram produzidas por A e B. K_S é a chave de sessão criada pelo KDC e distribuída conforme as mensagens 3 e 4 da figura 2.6. Ao criptografar a chave de sessão com as chaves secretas de A e de B nas mensagens 3 e 4, o KDC prova sua identidade, pois só ele, além dos proprietários das chaves conhece K_A e K_B. Quando A e B receberem as mensagens 3 e 4 só serão capazes de extrair a chave de sessão se possuírem de fato K_A e K_B, o que por si já serve para comprovar suas identidades [6].

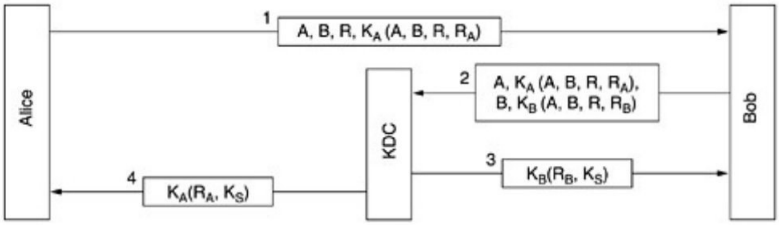

Figura 2.6: protocolo de autenticação Otway Rees [6].

2.4.3 Autenticação através de criptografia de chave pública

Este protocolo de autenticação é tão confiável quanto o sistema de distribuição de chave pública por ele adotado, por isso é fundamental usar uma infraestrutura que permita distribuir e validar chaves públicas de maneira segura. A figura 2.7 será utilizada para explicar o processo de autenticação com criptografia de chave pública.

No protocolo ilustrado na figura 2.7, a entidade A envia uma solicitação para estabelecer comunicação segura com a entidade B por meio da mensagem 3: $E_B(A, R_A)$. Antes de enviar a mensagem 3, A obteve a chave pública de B com uma terceira parte, considerada confiável, através das mensagens 1 e 2. Uma vez que A confia ter obtido a chave pública verdadeira de B, considerará a entidade B legítima se ela conseguir decifrar a mensagem 3, pois esta mensagem foi criptografada com a chave pública de B e somente

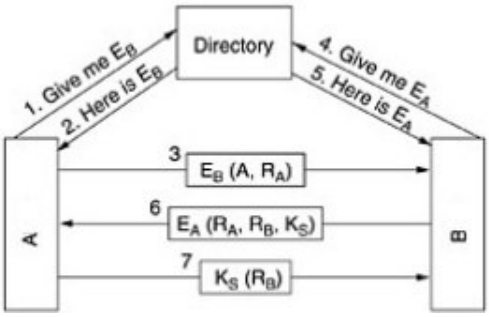

Figura 2.7: protocolo de autenticação mútua com criptografia de chave pública [6].

B, proprietária da chave privada correspondente, será capaz de desvendá-la. Quando B decifra a mensagem 3, percebe que se trata de solicitação de conexão segura proveniente de A, assim, logo procura obter a chave pública de A consultando uma terceira entidade de sua confiança através das mensagens 4 e 5. Na mensagem 6, $E_A(R_A, R_B, K_S)$, B responde ao desafio R_A, lançado na mensagem 3, e também envia um desafio a A, R_B, junto com uma chave de sessão, K_S.

A autenticação mútua estará concluída se A conseguir decifrar a mensagem 6 e retornar como resposta a mensagem 7 ($K_S(R_B)$), composta pelo desafio R_B criptografado com a chave de sessão K_S proposta por B. A entidade A só conseguirá continuar a comunicação com B, fazendo uso da chave de sessão, se possuir a chave privada correspondente a chave pública com que a chave de sessão foi criptografada.

É importante observar que este protocolo de autenticação pode ser executado parcialmente, neste caso apenas um dos lados é autenticado, como ocorre na autenticação via protocolo SSL, em que pode ser exigido apenas do servidor autenticação com chave pública. A etapa de obtenção da chave pública, que nesta sessão não ficou bem definida, pode ser solucionada através do gerenciamento de chave pública, assunto abordado na sessão 2.3 deste capítulo. Outro aspecto a ser destacado é que as chaves públicas empregadas no protocolo promovem a confidencialidade e a autenticação das mensagens trocadas.

2.5 Assinatura Digital

Com a criptografia de chave simétrica e assimétrica é fácil obter confidencialidade de mensagens, basta criptografá-las com a chave de sessão estabelecida ou com a

chave pública de quem vai recebe-las, prevenindo a troca de mensagens contra o acesso indevido de terceiros. Entretanto, além de confidencialidade e autenticação, um canal seguro de comunicação também deve garantir integridade de mensagens, propriedade que permite verificar se o conteúdo de uma mensagem foi indevidamente alterada durante a transmissão [1].

A assinatura digital é a técnica criptográfica desenvolvida para adicionar integridade de mensagem a um canal seguro de comunicação, com ela é possível verificar se o conteúdo transmitido permanece autêntico, qualquer mudança no conteúdo da mensagem assinada terá como resultado um valor de assinatura completamente diferente daquele obtido com a assinatura da mensagem original. Junto a integridade, a autenticação e a irretratabilidade são propriedades inerentes a assinatura digital: cada assinatura está relacionada de maneira única a identidade daquele que a gerou e uma vez produzida, a assinatura não poderá ser negada. Em [6] o autor classifica assinatura digital em dois grupos: assinatura de chave simétrica e assinatura de chave pública.

2.5.1 Assinatura de Chave Simétrica

Este processo de assinatura digital depende de uma estrutura formada por um centro de distribuição de chaves (KDC), a quem os usuários encaminham as mensagens que precisam ser assinadas. A figura 2.8 representa o processo de assinatura digital de chave simétrica, neste processo a mensagem 1 é uma solicitação de envio de mensagem assinada do usuário Alice ao usuário Bob através do centro de distribuição de chaves BB. Por meio da mensagem 1 Alice pede a BB que assine a mensagem P e a envie ao usuário Bob. Os parâmetros A, K_A, B, R_A, t e P, significam, respectivamente, o identificador de Alice, a chave secreta compartilhada entre Alice e o KDC BB, o identificador de Bob, o número aleatório escolhido por Alice, o carimbo de tempo (timestamp) e a mensagem em texto simples.

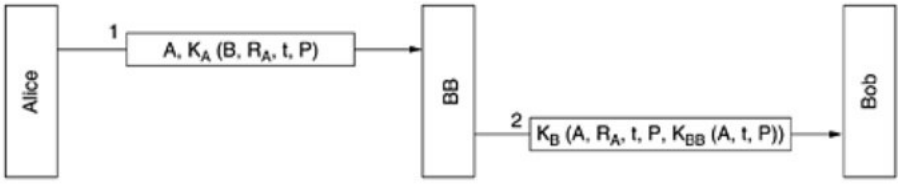

Figura 2.8: assinatura digital com chave simétrica [6].

Ao receber a mensagem 1, BB identifica seu remetente e a descriptografa com a chave compartilhada com ele, em seguida verifica se a mensagem não é uma repetição e sua atualidade através de R_A e t. Se a mensagem 1 for autêntica e atual, BB a assina, $K_{BB}(A, t, P)$, e repassa, confidencialmente, a Bob o conteúdo da mensagem original junto com a assinatura por meio da mensagem 2. Quando Bob receber a mensagem 2 saberá que ela só pode ter vindo de BB, com quem compartilha K_B, e poderá analisar sua integridade, basta criptografar parte do seu conteúdo com a chave K_{BB} e comparar com a assinatura enviada. Na assinatura, a presença da identidade de Alice serve para provar a origem da mensagem. A confiança de todos no KDC, junto com a identidade A e o carimbo de tempo t tornam a assinatura incontestável. O problema nesta abordagem está no fato de todos terem de confiar no KDC e de todas as mensagens serem lidas pelo KDC.

2.5.2 Assinatura de Chave Pública

Este tipo de assinatura digital consiste em criptografar com a chave privada do signatário uma mensagem ou um resumo de mensagem (hash), quem tiver acesso a chave pública complementar será capaz de recuperar a mensagem ou resumo de mensagem original. Quando o conteúdo assinado é descriptografado com a chave pública do signatário, está comprovada a autenticidade da mensagem, pois somente o dono da chave privada correspondente poderia tê-la produzido. Além disso, como apenas o dono da chave privada tem acesso a ela, não poderá negar sua assinatura posteriormente, propriedade conhecida como não repudiação ou irretratabilidade. Se a mensagem original for enviada junto com a assinatura, então será possível verificar sua integridade basta comparar a mensagem original ou seu resumo com a mensagem ou resumo recuperado a partir da assinatura, se os valores obtidos forem idênticos, então estará comprovada a integridade da mensagem original.

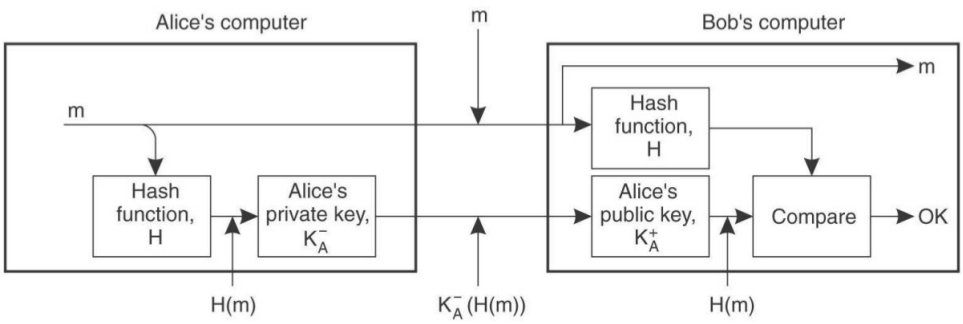

Figura 2.9: assinatura de chave pública [1].

A figura 2.9 representa o processo de assinatura com a criptografia de chave assimétrica. Na representação, Alice usa sua chave privada K_A^- para criptografar o resumo de mensagem H(m), o que resulta na assinatura $K_A^-(H(m))$. Alice envia a mensagem original, m, e a assinatura, $K_A^-(H(m))$, a Bob, que, ao receber o conteúdo enviado, aplica a chave pública de Alice sobre a assinatura gerada por ela e recupera o resumo de mensagem H(m). Ao mesmo tempo, Bob aplica uma função hash sobre a mensagem m, agora é possível comparar os resultados obtidos, se forem iguais, então m é íntegra e foi, inegavelmente, produzida por Alice.

No processo ilustrado a assinatura foi produzida sobre o resumo de mensagem H(m), o que é bastante comum, pois assinar a mensagem m inteira é computacionalmente custoso. Outro aspecto a ser observado é o fato de toda a comunicação estar ocorrendo de forma não confidencial. Se quisesse proteger o conteúdo enviado, Alice teria que criptografar pelo menos a mensagem m com a chave pública de Bob. Além do mais, o algoritmo de criptografia de chave pública usado neste tipo de assinatura deve possuir as seguintes características para justificar seu uso nesta finalidade [6]:

1. O algoritmo de criptografia e descriptografia pode ser aplicado em qualquer ordem, $K_A^-(K_A^+(m)) = K_A^+(K_A^-(m)) = m$, o que torna a assinatura facilmente verificável.

2. Impossível haver m e m', com $m \neq m'$, tal que $K_A^-(m') = K_A^-(m)$, ou seja, é impossível falsificar uma assinatura.

A assinatura de chave pública depende fortemente do sigilo mantido sobre a chave privada, se esta chave for violada não há garantia alguma em relação a integridade, irretratabilidade e autenticação das mensagem assinadas com ela. A solução largamente utilizada para resolver problemas desta natureza é o gerenciamento de chaves públicas, assunto abordado na sessão 2.3, onde uma ou várias autoridades certificadoras disponibilizam certificados de chave pública, que comprovam a autenticidade de cada chave pública divulgada, e listas de certificados revogados, que servem para informar quais certificados de chave pública não devem ser mais utilizados devido a invalidade da chave privada correspondente.

2.6 Autorização

Os conceitos de criptografia, autenticação, assinatura digital e gerenciamento de chave apresentados até o momento neste capítulo são fundamentais para a comunicação segura. Nesta sessão outro conceito fundamental a segurança de sistemas computacionais será abordado, trata-se do mecanismo de autorização, muito importante para impor uma política de segurança e, mais especificamente, o controle de acesso.

Um canal seguro de comunicação garante autenticidade, integridade e confidencialidade, uma vez asseguradas estas propriedades o passo seguinte é verificar se as requisições enviadas através do canal seguro são permitidas ou não. A política de segurança de um sistema computacional seguro define todos os requisitos de segurança do sistema, um conjunto de regras onde são definidas todas as ações, as entidades do sistema e as relações entre as ações que cada entidade pode executar.

Autorização é a concessão de direitos de acesso a recursos de um sistema, ao passo que controle de acesso é a ação de verificar se a entidade que faz uma requisição tem de fato direito de acesso ao recurso pretendido. O controle de acesso pode ser implantado através de Firewalls, matriz de controle de acesso e domínios de proteção, por exemplo [1]. No entanto, as diferentes formas de implementação do controle de acesso frequentemente seguem o modelo geral de controle de acesso, já explicado no capítulo anterior e representado através da figura 1.1. Neste arquétipo há três componentes principais: sujeito, objeto e monitor de referência. O sujeito é a entidade que faz requisições, o objeto é o recurso alvo das requisições e o monitor de referência é o componente que armazena informações de direitos de acesso.

2.6.1 Matriz de Controle de Acesso

A matriz de controle de acesso é um modelo que implementa o monitor de referência como uma estrutura de dados cujas linhas representam os sujeitos e as colunas os objetos. Sendo M a matriz, cada célula M[s, o] armazena informações sobre quais métodos de um determinado objeto o um sujeito s tem permissão para acessar. Neste modelo, o controle de acesso reside em verificar se o sujeito s que faz uma requisição através da chamada a um método m do objeto o tem de fato direito de acesso a este método. Assim o controle de acesso se resume a consultar células da matriz.

A desvantagem desta abordagem está na quantidade de entradas vazias que pode deixar na matriz, resultando em desperdício de recursos como espaço de armazenamento de dados, por exemplo. Para reduzir desperdício de recursos a matriz de controle de acesso pode ser dividida em várias colunas, cada objeto passa a manter uma coluna, formando uma lista conhecida como **lista de controle de acesso** (Access Control List - **ACL**). A ACL associada a um objeto só tem entrada para sujeitos que tem direito de acesso a ele. Cada entrada da lista informa quais métodos do objeto um sujeito tem direito de acessar.

A matriz de controle de acesso também pode ser redistribuída em linhas, cada linha é associada a um sujeito e é conhecida como lista de **capacidades**. As entradas da lista de capacidades mantida por cada sujeito indicam os métodos dos objetos que ele tem direitos de acesso.

A figura 2.10 representa dois modelos de controle de acesso. Em *(a)* o objeto controlado pelo servidor está associado a uma ACL. Quando o servidor recebe uma requisição r enviada pelo sujeito s direcionada ao objeto por ele controlado repassa a verificação de direito de acesso ao monitor de referência, este consulta a ACL associada ao objeto. A consulta consiste em analisar se o sujeito s aparece em alguma entrada da ACL, caso apareça, o monitor verifica se a requisição r está entre os métodos permitidos a s.

Na figura 2.10 (b), o controle é feito através de lista de capacidades. Neste modelo o sujeito envia uma requisição ao objeto protegido pelo servidor, uma capacidade é anexada à requisição. O monitor de referência do servidor apenas verifica se a capacidade é válida, ou seja, se o método do objeto o chamado através da requisição r aparece dentro da lista de capacidades C do sujeito. Assume-se que a lista de capacidades é inviolável, nem mesmo seu proprietário poderá alterá-la. Existem várias abordagens sobre a proteção de listas de capacidades, uma delas é o uso de certificados de atributos.

2.6.2 Domínios de Proteção

Listas de controle de acesso e de capacidades podem se tornar muito longas, inviabilizando a eficiência destes dois modelos. Uma alternativa a estas duas implementações é o modelo de domínios de proteção, com ele é possível implantar autorização de forma mais generalista, com menos dependência de informações de identidade do su-

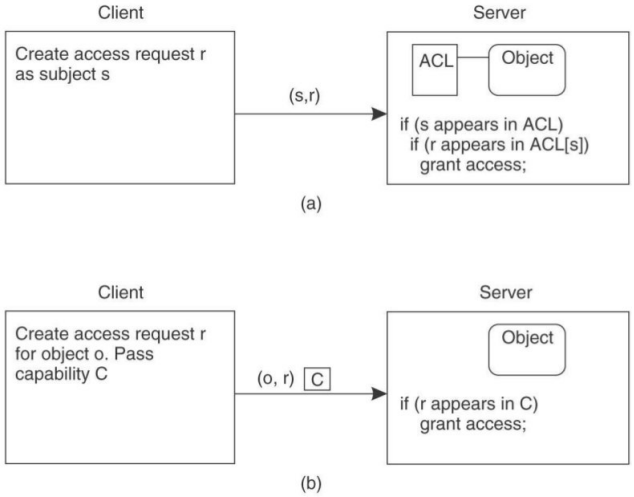

Figura 2.10: funcionamento do modelo de controle (a) ACL e (b) lista de capacidades [1].

jeito e do objeto. Neste modelo o sujeito pode ser identificado pelo grupo ao qual pertence ou pelo papel que desempenha dentro de uma organização, por exemplo. Objetos podem ser identificados por meio das interfaces das classes a que pertencem, ao invés de um identificador particular.

Um *domínio de proteção* é um conjunto de pares *objeto, direitos de acesso*. Em cada par do domínio de proteção, o componente direitos de acesso descreve as operações do objeto que podem ser executadas e por quem. Usando referências a classes de interface do sistema e grupos ou papeis, as regras de direito de acesso podem ser definidas através de declarações que registram os métodos das interfaces que podem ser acessados e os grupos ou papeis que tem direito de acessá-los. A lista de controle de acesso fica implícita no mapeamento entre grupos e métodos de interface permitidos a eles. Nesta forma particular de abordar domínios de proteção, quando uma requisição chega ao servidor, o monitor de referência identifica a que interface(s) pertence o objeto alvo da requisição. Tendo identificado o objeto, ou melhor, a(s) interface(s), o monitor de referência verifica se no domínio do objeto há algum mapeamento dentro das regras de direito de acesso que relacione a operação requisitada e o(s) grupo(s) do sujeito que fez a requisição.

Organizar sujeitos em grupos e objetos através herança de interface facilita o gerenciamento de segurança, cadastrar um sujeito em um grupo significa adicionar uma série de permissões a ele de uma vez só. Esta flexibilidade pode ser potencializada ainda mais se grupos de usuários forem organizados hierarquicamente. Se os usuários portarem certificados que declaram os grupos a que estão associados, o controle de acesso realizado pelo monitor de referência fica ainda mais simples, pois elimina a busca pelos grupos do

usuário antes da consulta às permissões. Neste caso, o certificado desempenha a mesma função de uma lista de capacidades.

O gerenciamento de direitos de acesso pode ser melhorado ainda mais através do uso de capacidades, certificados de atributo e técnicas de delegação. Como foi visto na subseção 2.6.1, a lista de capacidades é uma das formas de implementar uma matriz de controle de acesso, nela a capacidade é o componente base que consiste numa estrutura de dados onde são armazenadas informações sobre todas as operações que um sujeito pode executar sobre um recurso específico, esta estrutura deve ser à prova de falsificação. Sempre que um cliente quer acessar um recurso protegido, ele passa a capacidade correspondente ao componente que protege o recurso, este componente deve dispor de mecanismos para verificar a validade da capacidade. Certificados de atributos podem ser usados na validação de capacidades. Neste tipo de certificação uma autoridade certificadora especial assina as capacidades atribuídas a um sujeito, dando validade aos direitos de acesso associados aos recursos identificados no certificado do sujeito.

Outra forma de melhorar o gerenciamento de segurança é através da delegação de direitos de acesso. Neste modelo de gerenciamento de direitos de acesso um processo pode repassar parte de seus direitos a outro processo quando necessário. Existem várias abordagens para implementar delegação: um conjunto de direitos pode ser repassado através de **proxy**, por exemplo. Um proxy é um token, que contém um subconjunto de direitos que permitem ao seu portador atuar com poder igual ou inferior ao do processo que concedeu os direitos. Este token pode ser implantado sob forma de certificado digital, por exemplo, composto de uma chave pública criada especificamente para o proxy e a declaração dos direitos que estão sendo concedidos, este certificado é assinado pelo processo cedente. O processo cedente e o processo delegado compartilham a chave privada criada para o token, este segredo compartilhado é usado pelo processo delegado para provar que recebeu o conjunto de direitos descrito no certificado. Este esquema de delegação, com uso de proxy e certificado de atributos, é conhecido como abordagem de Neuman, e está representado na figura 2.11. O certificado é representado por $[R, S^{+}_{proxy}]_A$: R é o conjunto de direitos de acesso e S^{+}_{proxy} é a chave pública do proxy. Todo o conteúdo do certificado é assinado por Alice (A).

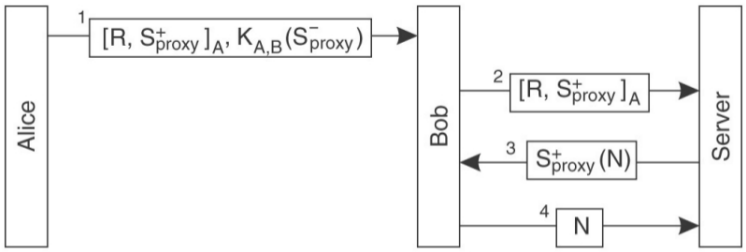

Figura 2.11: delegação através de proxy e certificado [1].

2.7 Auditoria

No contexto de segurança de sistemas computacionais, auditoria é o armazenamento inviolável de dados sobre eventos relacionados a segurança com o intuito de detectar a efetividade da política e dos mecanismos de segurança adotados [3]. Dentre os mecanismos de segurança já apresentados neste capítulo, a auditoria é o que parece estar menos relacionado a segurança. No entanto, a simples ação de registrar as operações realizadas em um sistema pode favorecer a identificação de falhas, a adoção de medidas de reação contra ataques e inibir a ação de atacantes, pois torna as tentativas de burlar a segurança mais cautelosas [1].

É importante observar que o bom funcionamento dos demais mecanismos é crucial para a efetividade da auditoria. Se a autenticação não estiver funcionando corretamente, então não há como ter certeza se as transações executadas em nome de um usuário foram de fato realizadas por ele. Pior ainda se o atacante burlar a autenticação, pois poderá passar pelo sistema sem deixar registro algum que leve a sua identidade. Em caso de falhas na autorização, o próprio registro de dados de relevantes a auditoria poderá ficar exposto a alterações por usuários não autorizados. Se os registros de auditoria forem protegidos por criptografia, os dados do registro estarão protegidos contra modificação e interceptação, por exemplo.

2.8 Conclusão

O presente capítulo abordou conceitos de segurança, criptografia, autenticação e gerenciamento de segurança para formar a base teórica necessária a compreensão da solução de segurança por trás da tecnologia Java EE e do sistema SIAP. Os vários métodos de autenticação, as diferentes abordagens sobre autorização, as diversas técnicas de crip-

tografia e assinatura digital foram alguns dos assuntos delineados para que os próximos capítulos possam ser entendidos.

3 Tecnologias Utilizadas

O Sistema Integrado de Apoio ao Paciente (SIAP) foi desenvolvido com a tecnologia Java EE 5, usando especificamente as tecnologias JavaServer Faces (JSF), Enterprise Java Beans (EJB), Java Persistence API (JPA), Java Persistence Query Language e os serviços de Segurança e Transação da plataforma Java EE. A escolha da tecnologia Java EE para desenvolver o sistema SIAP ocorreu devido ao simplificado modelo de programação que ela apresenta e as facilidades que oferece para a criação de aplicações distribuídas, portáveis, seguras, confiáveis, menos complexas e de alta performance [3].

Aplicativos Java EE são escritos em linguagem de programação Java e são projetados para implementar serviços que podem precisar acessar várias fontes de dados ou aplicações distribuídas e servir uma infinidade de clientes ao mesmo tempo, o que pode levar certa complexidade ao projeto de um aplicativo. Para suportar aplicações com estas características, o modelo de desenvolvimento Java EE se embasa numa arquitetura organizada em múltiplas camadas. Esta arquitetura traz um conjunto de serviços primários que permitem o desenvolvedor voltar seus esforços para a lógica de apresentação e de negócio do sistema, deixando para a arquitetura soluções de problemas difíceis em nível de desenvolvimento multi-camadas, tais como escalabilidade, acessibilidade, gerenciabilidade e segurança [3].

O capítulo anterior tratou da fundamentação teórica, introduzindo os conceitos necessários a apresentação da solução de segurança adotada pelo SIAP. O presente capítulo irá abordar as tecnologias que foram empregadas na construção do sistema, descrevendo brevemente o modelo e a arquitetura de uma aplicação Java EE, expondo os tipos de componentes que constituem o sistema e apresentando as ferramentas necessárias a implementação das funcionalidades de segurança.

3.1 O Modelo de Componentes

Sistemas corporativos escritos de acordo com a especificação Java EE tem sua lógica de negócio implementada como um conjunto de pequenas unidades de software

chamadas de componentes. Em [3] um componente é definido da seguinte maneira:

> Um componente Java EE é uma unidade funcional de software auto-suficiente que reunida com suas classes e arquivos relacionados se comunica com outros componentes e integra uma aplicação Java EE.

Estas unidades funcionais são escritas em linguagem de programação Java, implantadas em servidores Java EE e gerenciadas por eles. Uma das principais características do modelo baseado em componentes é a independência de implementação, isto significa que um componente pode executar em qualquer servidor de aplicação que respeite as determinações da especificação Java EE, tais como os servidores JBoss, GlassFish, WebLogic, etc. Componentes são formados por objetos distribuídos e podem ser instalados em diferentes máquinas dependendo da camada da arquitetura a qual pertença [2].

A figura 3.1 apresenta os diferentes tipos de componentes distribuídos nas suas respectivas camadas. Nesta figura há dois aplicativos multi-camadas, um aplicativo cliente e um aplicativo web, estes aplicativos podem estar divididos em quatro camadas, conforme ilustra a figura 3.1:

- Camada Cliente, os componentes desta camada se localizam na máquina cliente e fazem parte da lógica de apresentação. Geralmente usuários interagem com as aplicações que ficam nesta camada, que por sua vez interagem com componentes das demais camadas, repassando dados de entrada ou apresentando resultados provenientes delas. Os componentes Java EE típicos desta camada são: applets e aplicações cliente, que serão explicados a frente;

- Camada Web, os componentes desta camada se localizam no servidor Java EE e possuem componentes da lógica de apresentação e de negócio. A lógica de apresentação está nas páginas web, por exemplo, que apesar de serem processadas do lado do servidor, produzem como resultado páginas que são exibidas no cliente web. A lógica de negócio está em componentes JavaBeans que interagem com as camadas abaixo. São componentes Java EE típicos desta camada Java Servlet, JavaServer Faces e JavaServer Pages;

- Camada de Negócio, os componentes desta camada se localizam no servidor Java EE, são voltados para a lógica de negócio e podem interagir com componentes das demais

camadas. São componentes Java EE típicos desta camada os EJBs (Enterprise Java Beans);

- Camada de Armazenamento de Dados (Enterprise Information System - EIS), os componentes desta camada se localizam no servidor EIS e tem como principal função gerenciar a persistência de dados.

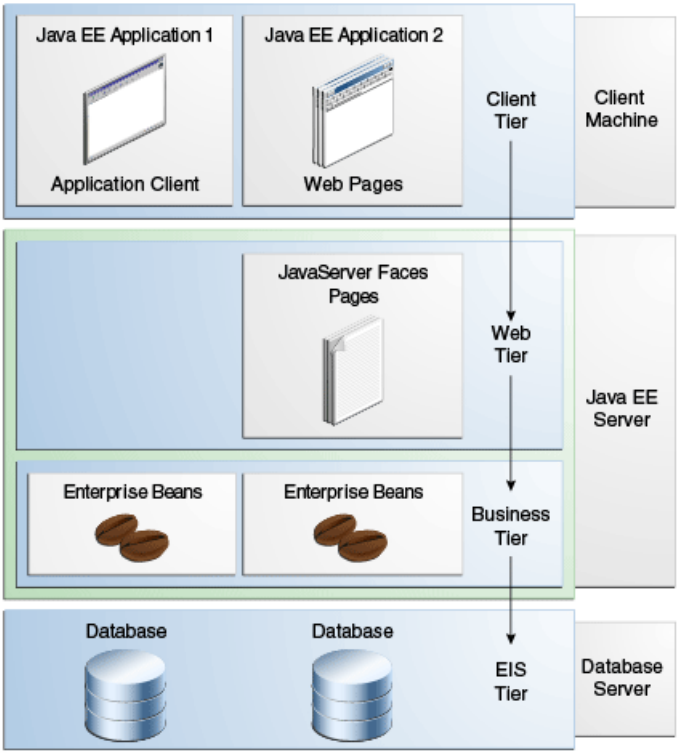

Figura 3.1: modelo de aplicação em múltiplas camadas [3].

3.2 A Arquitetura de uma Aplicação Java EE

A especificação Java EE tem como padrão um modelo de desenvolvimento em camadas baseado em componentes [3]. Existem vários tipos de componentes, cada um deles segue um padrão também determinado pela especificação. Nesta sessão veremos a arquitetura que foi projetada para facilitar o desenvolvimento de aplicações que obedeçam o modelo multi-camadas proposto.

Um conceito importante introduzido pela arquitetura é o **contêiner**, que pode ser entendido como um conjunto de serviços provido pelo servidor Java EE para os componentes de um aplicativo. O contêiner é uma interface entre componentes e funcionalidades

oferecidas pela plataforma a cada tipo de componente, ele dispensa o desenvolvedor de voltar seus esforços para problemas da arquitetura em múltiplas camadas [3].

Um aplicativo Java EE pode ser formado por vários tipos de módulos: módulo de aplicação cliente, módulo de aplicação web e módulo EJB, por exemplo. Cada módulo é implantado num contêiner e envolve um conjunto de componentes típicos, um módulo web reúne apenas componentes web, por exemplo. Em cada módulo há informações de configuração que servem para informar ao contêiner como os serviços da plataforma subjacente serão utilizados pela aplicação. As informações de configuração customizam o contêiner para que ele ofereça os serviços do servidor Java EE voltados para as necessidade específicas da aplicação. Cada servidor de aplicação Java EE implementa ao seu modo o conjunto de serviços primários da especificação Java EE. Sendo o contêiner uma interface, os componentes podem usar serviços de segurança, gerenciamento de transação, conectividade remota, localização e nomeação de objetos, por exemplo, sem se importar com a forma que eles são implementados pelo servidor de aplicação [3]. A seguir, cada uma das camadas será apresentada, junto com os componentes típicos de cada uma delas.

3.2.1 A Camada Cliente

Esta camada fica na máquina cliente e executa **clientes web**, **aplicações cliente** ou **applets** [3]. Falaremos em sequência de cada um destes componentes.

Um **clientes web** é um navegador que exibe as páginas recebidas do servidor, estas páginas são dinâmicas e podem conter elementos escritos vários tipos de linguagem de marcação (html, xhtml, etc.). As páginas são geradas por componentes web do lado do servidor que costumam executar tarefas mais pesadas, como operar regras de negócio ou consultar banco de dados, por exemplo.

A **aplicação cliente** é o tipo de componente voltado para manipular tarefas que exigem interfaces gráficas mais robustas. Ela pode interagir diretamente com componentes de negócio ou com componentes web. Também é possível haver aplicações cliente escritas em outras linguagens além de Java, ou mesmo uma aplicação cliente ser parte de um sistema legado que interage com o servidor Java EE.

Um **Applet** é uma pequena aplicação cliente embutida numa página web. A execução de um applet é mais complexa que a de uma página em um cliente web, pois um

applet precisa que a máquina virtual Java esteja instalada no navegador e de permissões especiais para acessar o sistema de arquivos local, por isso, sempre que possível, é mais recomendável o uso de aplicação cliente, que não exige configurações especiais.

Na camada cliente pode haver componentes JavaBeans, estes são objetos Java simples com atributos e métodos get e set. Eles não são considerados componentes Java EE, mas podem gerenciar o fluxo de dados entre componentes da camada cliente e do servidor. Uma aplicação cliente é envolvida por um contêiner que gerencia os componentes da aplicação, permitindo que eles se comuniquem remotamente com componentes do lado do servidor de forma transparente.

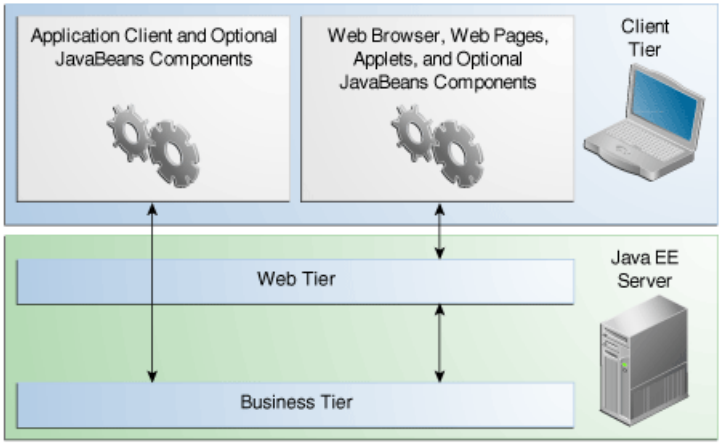

Figura 3.2: comunicação com a camada cliente [3].

A figura 3.2 apresenta a interação entre os mais diversos tipos de componentes da camada cliente com componentes do servidor Java EE. Um elemento da camada cliente pode interagir direto com a camada de negócios ou interagir com esta camada por intermédio de componentes da camada web.

3.2.2 A Camada Web

Os componentes desta camada são Servlets ou páginas web criadas com as tecnologias JavaServer Faces (JSF) ou JSP. As páginas HTML exibidas num **cliente web** são resultado das páginas produzidas pelos componentes web que executam no servidor Java EE. Pode haver componentes JavaBeans na camada web, neste caso JavaBeans são úteis para gerenciar dados de entrada das páginas e repassa-los a camada de negócio.

A figura 3.3 apresenta navegador, páginas web, applets e componentes Java

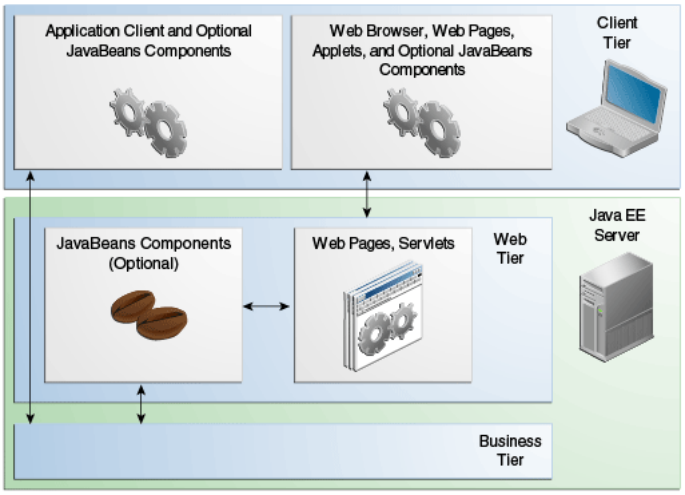

Figura 3.3: camada web [3].

Beans na camada cliente interagindo com componentes web. Um componente web pode fazer uso de componentes JavaBeans para repassar dados recebidos do cliente a camada de negócio.

3.2.3 A Camada de Negócio

Nesta camada concentra-se toda a lógica de negócio do sistema. Existem dois tipos de componentes de negócio: Enterprise Beans, que implementam a tecnologia Enterprise Java Beans (EJB), e Beans de Entidade, ou Entidades de Persistência Java. Enterprise Beans são componentes do lado do servidor que encapsulam a lógica de negócio da aplicação, são classificados em dois tipos: beans de sessão ou beans baseados em mensagem (message-driven bean) [3].

Beans de sessão gerenciam processos ou tarefas de negócio para o cliente. Beans baseados em mensagem agem como ouvintes de tipos específicos de mensagens assíncronas para executar suas tarefas. Enquanto beans de sessão interagem com clientes por meio de invocações de métodos em suas interfaces, beans baseados em mensagem recebem mensagens e processam tarefas, sem retornar resposta, o cliente não fica a espera de um resultado. Os beans baseados em mensagem podem funcionar como ponto de integração com outros aplicativos ou sistemas legados. Beans de entidade, apesar de fazerem parte da camada de negócio, não são considerados componentes Java EE, eles podem ser adicionados a aplicativos Java que não seguem a especificação Java EE. Os beans de entidade são objetos do domínio de persistência, geralmente representam tabelas do banco de dados

e cada instância de entidade é uma linha da tabela [2].

Os componentes da camada de negócio recebem e processam dados da camada cliente ou web, interagindo com outros componentes de negócio ou com a camada de dados abaixo. A figura 3.4 apresenta os componentes de negócio interagindo com as camadas web e de dados. Os beans de negócio são envolvidos pelo contêiner EJB, por isso podem usar os serviços primários da plataforma.

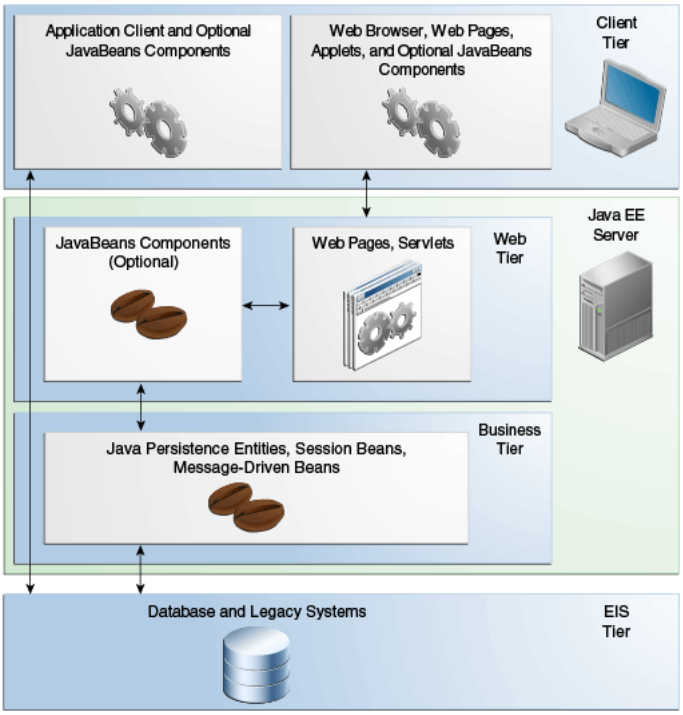

Figura 3.4: componentes da camada de negócio [3].

3.2.4 Camada de Armazenamento de Dados (Enterprise Information System - EIS)

Esta camada manipula a persistência de dados do sistema. Os componentes desta camada podem ser um sistema de gerenciamento de banco de dados ou um sistema legado, por exemplo [3]. Nas figuras 3.2, 3.3 e 3.4 é possível ver as diversas formas de interação entre cada uma das camadas já apresentadas com a camada EIS.

3.3 Segurança de Aplicação em Java EE

A arquitetura Java EE tem a sua disposição um conjunto de serviços de segurança providos pela plataforma subjacente. Um servidor Java EE oferece várias formas de incluir autenticação num aplicativo: através do fornecimento de credenciais de nome de usuário e senha, autenticação mútua por meio de apresentação de certificados digitais, dentre outros. Autorização é outro mecanismo que já vem incluído na plataforma, podendo ser adicionado facilmente a qualquer aplicativo Java EE. Estes dois mecanismos são baseados no framework JAAS (Java Authentication and Authorization Service), uma extensão que faz parte da plataforma padrão Java SE e que permite implementar autenticação e controle de acesso em qualquer aplicativo Java de forma versátil e extensível. Além do JAAS as seguintes APIs de segurança que fazem parte da plataforma Java SE são utilizadas pela plataforma Java EE [3]:

- Java Generic Security Services (Java GSS-API): adiciona troca de mensagens segura entre aplicações instaladas no cliente e no servidor.
- Java Cryptography Extension (JCE): provê implementações e frameworks para algoritmos de criptografia e gerenciamento de chaves.
- Java Secure Sockets Extension (JSSE): provê uma implementação para os protocolos SSL (Socket Security Layer) e TLS (Transport Layer Security), permitindo adicionar a um aplicativo integridade na troca de mensagens, encriptação de dados e autenticação do servidor, bases para a comunicação segura.
- Simple Authentication and Security Layer (SASL): provê uma implementação para o padrão da internet SASL (RFC 2222). Este padrão define como dados de autenticação devem ser trocados, sem especificar o formato dos dados e das mensagens trocadas, ele também delineia o estabelecimento de uma camada de segurança entre uma aplicação cliente e uma aplicação no servidor, que é opcional .

Todas estas APIs de segurança são utilizadas pela plataforma de maneira transparente para os aplicativos dentro do servidor, em conjunto, estas APIs adicionam segurança na camada de aplicação e na camada de transporte. Os contêineres do servidor Java EE viabilizam o acesso aos serviços implementados pela plataforma, assim, incluir

segurança numa aplicação Java EE pode ser algo bastante configurável, sendo possível aplica-la através da técnica programática ou declarativa.

A técnica declarativa é aplicada por meio da inclusão de anotações específicas nas classes dos componentes que formam a aplicação ou através de metadados de segurança nos arquivos XML descritores de implantação da aplicação. A técnica programática é utilizada apenas quando a forma declarativa não consegue resolver as necessidades de segurança da aplicação, neste caso, há um conjunto de métodos disponibilizados por meio das classes de interfaces do contêiner que podem auxiliar na implementação de uma solução particular [3]. Em ambas situações, a segurança de cada aplicação é uma solução decorrente da política de segurança previamente estabelecida para ela, portanto, cada solução é uma particularidade.

3.3.1 Conceitos importantes

Para compreender melhor as estratégias de autenticação e autorização do modelo de segurança da arquitetura Java EE é necessário introduzir alguns conceitos.

Domínio - Realm

Um Realm, ou Domínio, é um banco de dados de usuários e grupos de usuários considerados válidos para uma ou mais aplicações controladas pela mesma política de autenticação. É um ambiente que possui uma política (estratégia de autenticação e autorização) bem definida à qual se submetem todas as aplicações implantadas neste ambiente [3].

Os recursos protegidos em um servidor podem ser organizados em vários espaços (domínios) de proteção, cada um com seu próprio esquema de autenticação e autorização, contendo coleções de usuários e grupos. Este espaço de proteção é um realm ou domínio. As regras que determinam os recursos protegidos que cada usuário ou grupo pode acessar são estabelecidas através de uma política bem definida. Só podem acessar os recursos de um espaço protegido usuários válidos que estejam de acordo com a política do domínio. Num mesmo servidor de aplicação pode haver vários espaços de proteção (domínios, realms).

Dadas as explicações, um realm pode entendido como um espaço de proteção

que armazena informações sobre um conjunto de usuários autorizados a acessá-lo. Dentro deste espaço são implantadas aplicações que manipulam recursos sensíveis, estas aplicações concordam com o método de autenticação e de autorização usado pelo espaço de proteção para verificar se o usuário que faz uma requisição pertence ao seu banco de dados de usuários válidos.

Servidores de aplicação podem vir configurados com alguns tipos predefinidos de realm, é o caso do servidor GlassFish, que possui o *file*, *admin-realm* e *certificate* realms prontos para serem editados com a adição de usuários e grupos. O *file realm* armazena dados de usuários e grupos em um arquivo local chamado keyfile. O *admin-realm* também armazena dados de usuários em um arquivo local do servidor (admin-keyfile), mas serve apenas para cadastrar administradores do próprio servidor de aplicação, os quais pertencem sempre ao mesmo grupo, o *asadmin*. O gerenciamento de usuários destes dois tipos de realm é feito através do próprio servidor de aplicação. Já o *certificate realm* serve apenas para armazenar certificados digitais no padrão X.509 de usuários considerados válidos, ele é administrado através de ferramenta capaz de gerenciar certificados, tal como o keytool, que já vem incluído no GlassFish [3]. A autenticação via *certificate realm* é feita por meio da verificação do atributo "common name" do certificado digital. *Certificate realm* é útil a aplicação da autenticação do cliente via protocolo HTTPS.

Outros tipos de realm também são possíveis, é o caso do LDAP, JDBC e PAM realms. No JDBCRealm os dados dos usuários e grupos estão registrados em um banco de dados relacional com tabelas e atributos pre-determinados, enquanto no LDAPRealm usuários e grupos estão registrados num servidor LDAP. O PamRealm permite integrar uma aplicação com o esquema de autenticação empregado por alguns sistemas operacionais através da API PAM (pluggable authentication module). Caso nenhuma das implentações de realm disponíveis satisfaça as necessidades da aplicação, o desenvolvedor pode criar um realm customizado (custom realm), para isso é necessário estender algumas classes e configurações do framework JAAS e adicionar ao servidor de aplicação.

Usuário

Um usuário é a identidade de um indivíduo ou de uma aplicação registrada no servidor. Usuários podem ser associados a grupos ou papeis.

Grupo

Um grupo é um conjunto de usuários classificados por traços comuns, tais como profissão, categoria de cliente, etc. Organizar usuários em grupo facilita o controle de acesso de um grande número de usuários. Grupos são registrados dentro de um domínio.

Papel - Role

Um papel, ou role, é um nome abstrato para a permissão de acessar um conjunto particular de recursos em uma aplicação, é um agrupamento lógico de usuários segundo as funções que desempenham. Papeis de segurança são usados para definir a lógica de segurança do ponto de vista da aplicação [3].

Não se deve confundir papeis de segurança com conceitos de grupo, usuário ou principal. Grupo faz parte do escopo de um domínio do servidor, enquanto papel faz parte do escopo de uma aplicação. Isto significa que papel é algo mais próximo da aplicação, algo que é definido dentro da própria aplicação e é resultado da política de segurança definida especificamente para a ela.

Numa aplicação Java EE os papeis podem ser usados para limitar o acesso a páginas web e a métodos dos componentes enterprise, estas restrições são definidas através de anotações ou de metadados nos arquivos descritores de implantação, onde se especifica exatamente quais papeis podem acessá-los. Os usuários considerados válidos dentro de um realm são registrados nele como pertencentes a determinados grupos. Para que o controle de acesso seja aplicado durante a execução de um aplicativo Java EE é necessário que haja um mapeamento dos papeis registrados nele para os grupos existentes no realm. A figura 3.5 representa as possíveis formas de mapeamento: de papel para grupo e de papel para usuário, ou melhor, de papel para o principal de cada usuário.

Sujeito - Subject

Um sujeito é uma entidade (usuário ou serviço) que se submete a um processo de autenticação. Após passar pela autenticação com sucesso ao sujeito são anexadas informações relacionadas a sua identidade.

Sujeitos podem ter vários dados de identidade, em termos de implementação, cada identidade é representada por um atributo do tipo Principal, que simplesmente

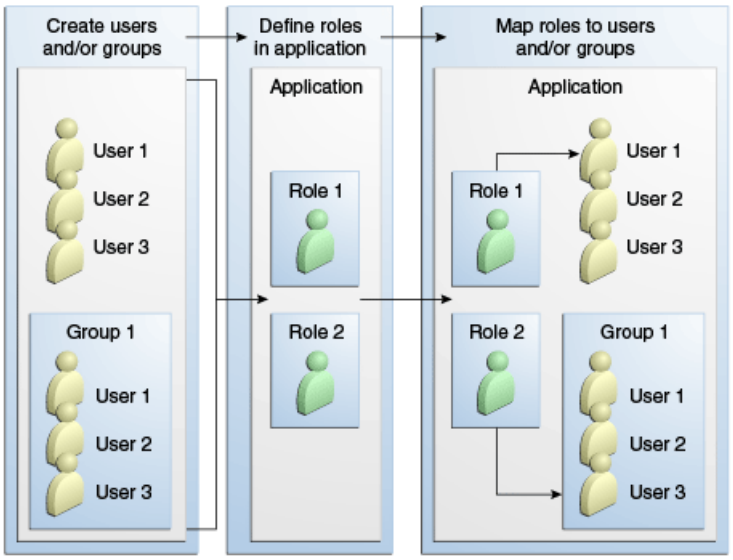

Figura 3.5: mapeamento de papel para usuários e grupos [3].

vincula a identidade ao sujeito. Um sujeito "pessoa", por exemplo, pode ter os atributos "nome completo" e "número de cpf" como principals. Os atributos de segurança de um sujeito são conhecidos como credenciais, que podem ser classificadas em dois tipos de credenciais: públicas (certificado digital, nome de usuário, por exemplo) e privadas (senha, chave privada, por exemplo) [13].

Principal

Um principal é um atributo de segurança constituído de informação capaz de identificar e representar o sujeito autenticado. As informações de identidade contidas nos Principals do sujeito autenticado ajudam a distingui-lo de outros sujeitos. Quando uma entidade é autenticada, a ela é atribuído um conjunto de principals que guarda suas identidades [13].

O diagrama 3.6 ilustra detalhes do funcionamento do framework JAAS durante o processo de autenticação, é possível observar que o sujeito só recebe instâncias de principals depois de passar pelo *login*.

Credencial - Credential

Uma credencial é um atributo relacionado a segurança que contem informação usada para autenticar o sujeito. Ela pode ser classificada como credencial pública ou privada. Senhas e certificados de chave pública são exemplo de credenciais do tipo privada

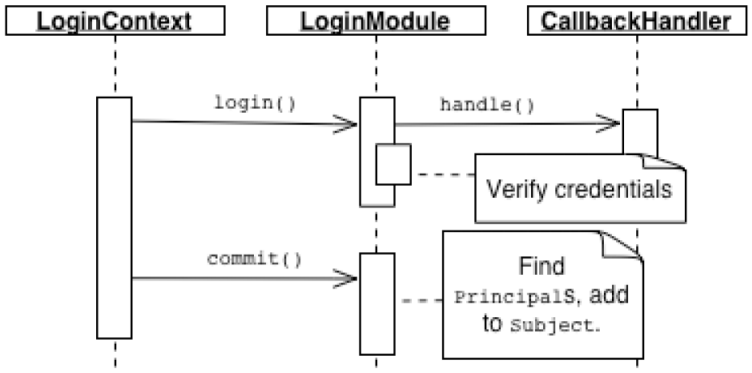

Figura 3.6: processo de login do framework JAAS [13].

e pública, respectivamente [13].

3.3.2 Autenticação

A autenticação é requisito que precede a autorização, em nosso modelo de segurança, a autenticação é feita através do fornecimento de nome de usuário e senha por meio de um formulário em página web, processo ilustrado na figura 3.7.

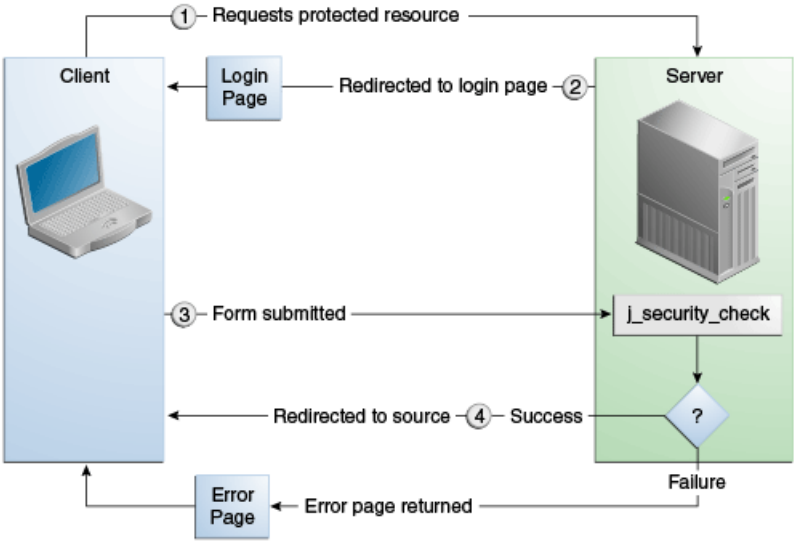

Figura 3.7: autenticação baseada em formulário [3].

Este método de autenticação é oferecido pela própria plataforma Java EE, que é capaz de autenticar das seguintes formas [3]:

- Autenticação básica: a autenticação é feita por meio do fornecimento de nome de usuário e senha através de uma caixa de diálogo enviada ao cliente web;

- Autenticação baseada em formulário: a autenticação também é feita através do fornecimento de nome de usuário e senha, a diferença em relação ao mecanismo anterior está no fato de os dados de autenticação serem coletados através de uma página web que pode ter sua aparência customizada pelo desenvolvedor;

- Autenticação digest: realiza o processo de autenticação da mesma forma que a autenticação básica, a diferença está no fato de enviar o hash da senha. Do lado do servidor a senha deve ser armazenada em texto claro, assim o servidor poderá gerar o hash da senha e comparar com aquele que foi enviado;

- Autenticação cliente: a autenticação ocorre via certificado de chave pública. O servidor verifica se o certificado enviado pelo cliente faz parte do seu banco de certificados considerados válidos para acessar o sistema. Os certificados devem estar no padrão X.509 e o atributo "common name" é usado como principal, ou seja, após passar pela autenticação, o "common name" atribuído ao sujeito é o atributo usada para verificar questões relacionadas a autorização. O *certificate realm* tem um conjunto de "common name" de usuários autorizados em seu banco de dados. Todo o processo de autenticação ocorre via protocolo HTTPS;

- Autenticação mútua: neste processo o cliente e o servidor autenticam um ao outro através de apresentação de certificados digitais ou apenas o servidor apresenta certificado enquanto o cliente é autenticado via nome de usuário e senha.

3.4 Conclusão

Este capítulo apresentou as tecnologias utilizadas para desenvolver o Sistema Integrado de Apoio ao Paciente, fazendo uma introdução as características da tecnologia Java EE, descrevendo seu modelo e arquitetura, além de dar destaque aos serviços de segurança da plataforma. Além da arquitetura e das funcionalidades de segurança Java EE, também foram descritas as APIs de criptografia da plataforma Java SE que foram empregadas na implementação do sistema e a finalidade delas, assunto que também será complementado no capítulo seguinte.

O sistema SIAP, objeto de estudo deste trabalho, foi implantado num servidor **GlassFish**, um servidor de aplicação que implementa todas as especificações da plataforma Java EE. Assim, o SIAP tem a sua disposição grande parte dos serviços e APIs

descritas neste capítulo. Dentre os serviços da plataforma, a segurança do servidor foi amplamente utilizada por meio da técnica declarativa. As funcionalidades de segurança que não puderam contar diretamente com o apoio da plataforma foram implementadas com recursos da plataforma Java SE. Detalhes sobre o uso das tecnologias apresentadas neste capítulo no sistema SIAP serão discutidos no próximo capítulo.

4 Modelo e Implementação dos Mecanismos de Segurança do Projeto SIAP

O Sistema Integrado de Apoio ao Paciente (SIAP) manipula informações de atendimento em saúde de pacientes, o que implica maior cuidado com o acesso, armazenamento e transmissão deste tipo de dado devido ao caráter sigiloso das informações que dizem respeito à saúde dos indivíduos. Preocupações desta natureza motivaram a criação de vários padrões, normas e leis para preservar o caráter sigiloso deste tipo de informação. Em nosso país, o Conselho Federal de Medicina (CFM), em parceria com a Sociedade Brasileira em Informática da Saúde (SBIS), desenvolveu um processo de certificação de sistemas informatizados de saúde com o objetivo homologar soluções que respeitem a legislação vigente e padrões internacionais que regulamentam este tipo de sistema [12].

O modelo de segurança do SIAP foi desenvolvido levando em consideração os requisitos de segurança para Sistemas de Registro Eletrônico em Saúde (S-RES) constantes no manual do referido processo de certificação. Com base nestes requisitos, nos serviços da plataforma Java EE e nas funcionalidades do SIAP foi estabelecida uma política de segurança para orientar a implementação dos mecanismos necessários. Este capítulo irá abordar o modelo de segurança do sistema SIAP (a central de atendimento web), tratando especificamente da política de segurança adotada e as primitivas de segurança usadas para impô-la.

4.1 Descrição geral do sistema

O sistema SIAP é fruto do projeto "Aplicações e Desenvolvimento Tecnológico em Telemedicina Baseado em Sistemas de Auxílio ao Diagnóstico" desenvolvido pelos laboratórios LESERC (Laboratório de Engenharia de Software e Rede de Computadores), PIB (Laboratório de Processamento da Informação Biológica) e LSD (Laboratório de Sistemas Distribuídos), os dois primeiros relacionados ao curso de Engenharia Elétrica e o último ao curso de Ciência da Computação, todos da Universidade Federal do Maranhão.

O projeto tinha por objetivo geral o desenvolvimento de ferramentas para auxílio ao diagnóstico de enfermidades cardíacas a serem executadas a partir de dispositivos móveis.

Pretendia-se, especificamente, fornecer um sistema eletrocardiográfico (ECG) móvel capaz de capturar sinais cardíacos de pacientes nas localidades mais remotas para posterior transmissão a uma central de dados, sendo necessário para isso, além do eletrocardiograma móvel, uma infraestrutura de software que permitisse aos especialistas (cardiologistas) o acesso as informações capturadas em campo e a emissão dos laudos correspondentes quando necessário.

A equipe do curso de Engenharia de Elétrica teve por responsabilidade desenvolver um equipamento de hardware capaz de monitorar e armazenar informações sobre o desempenho cardíaco do paciente, enquanto a equipe do LSD ficou responsável pelo desenvolvimento do sistema de controle dos dados do sistema (através de um Sistema Web) levando em consideração as primitivas de segurança exigidas para sistemas da área da saúde, desde básica autenticação e autorização até o tratamento de certificação digital, já que o sistema contempla a emissão de laudos médicos.

A relevância deste tipo de sistema está no fato de muitas vezes os pacientes das cidades do interior do estado não contarem com assistência de médicos especialistas, que costumam se concentrar nos grandes centros urbanos, além do fato de doenças do aparelho circulatório serem as principais causas de morte na população brasileira [14].

O sistema foi criado para prover importantes funcionalidades, tais como, a geração de prontuários eletrônicos de pacientes, visando facilitar o registro e o acesso às informações, sinais e imagens relacionados à saúde e à assistência. Dados relacionados a exames de ECG são coletados e enviados a uma equipe de especialistas do Hospital Universitário da Universidade Federal do Maranhão utilizando-se dispositivos móveis do tipo smartphone.

4.2 O manual de certificação SBIS/CFM

O Manual de Certificação para Sistemas de Registro Eletrônico em Saúde (S-RES) trata de questões de privacidade e confidencialidade dos indivíduos assistidos, à integridade e segurança das informações e aos recursos mínimos necessários para o perfeito registro dos atos praticados e das condições de saúde dos indivíduos [12]. O referido

manual responde questões concernentes à legalidade da utilização de sistemas informatizados para capturar, armazenar, manusear e transmitir dados do atendimento em saúde, incluindo as condições para a substituição do suporte papel pelo meio eletrônico e resulta de um convênio de cooperação técnica entre a Câmara Técnica de Informática em Saúde do Conselho Federal de Medicina (CFM) com a Sociedade Brasileira de Informática em Saúde com o objetivo de desenvolver o processo de certificação de sistemas informatizados em saúde.

Atualmente o manual se sencontra na versão 4.1, publicada em outubro de 2013, e detalha todas as etapas do processo de auditoria ao qual devem se submeter os S-RES que buscam a certificação. A Certificação SBIS/CFM se baseia em conceitos e padrões nacionais e internacionais da área de Informática em Saúde.

Para fins da Certificação SBIS-CFM, pode ser submetido ao processo qualquer S-RES que atenda minimamente à seguinte categoria:

- **Básica**: S-RES voltados à assistência à saúde de indivíduos, de forma básica e genérica (não específica);
- **Ambulatorial**: S-RES voltados para a assistência ambulatorial, tais como automação de consultórios clínicos, clínicas, unidades básicas de saúde, etc.

A certificação SBIS-CFM prevê ainda níveis diferenciados nos requisitos de segurança para os S-RES, o grupo de requisitos exigidos depende da categoria em que o sistema se enquadrar. O S-RES poderá ser enquadrado em dois níveis distintos de garantia de segurança: um primeiro nível (Nível de Garantia de Segurança 1 - NGS1) mais amplo e um segundo nível (Nível de Garantia de Segurança 2 - NGS2) que, além de contemplar todos os requisitos do primeiro nível, exige também que o S-RES incorpore as funcionalidades necessárias para que o sistema opere sem a geração de registros impressos (sistema sem papel - paperless) [12]. Adicionalmente, o S-RES deverá ser identificado como sendo um S-RES local ou remoto, refletindo se o mesmo funciona somente no próprio computador onde for instalado (local) ou se pode ser acessado remotamente a partir de estações de trabalho conectadas ao computador (remoto).

Para ser aprovado, um S-RES precisará necessariamente se enquadrar pelo menos na categoria Básica descrita acima, atendendo a todos os requisitos obrigatórios estabelecidos para a mesma, além de atender também a todos os requisitos obrigatórios

previstos pelo menos no Nível de Garantia de Segurança 1 - NGS1.

Caberá ao Solicitante indicar as categorias de sistema e o nível de garantia de segurança do seu S-RES, para que estas informações sejam consideradas no processo de certificação [12].

No total, todos os requisitos da certificação SBIS-CFM foram reunidos nos seguintes grupos:

- Requisitos de Segurança;
- Requisitos de Estrutura, Conteúdo e Funcionalidades;
- Requisitos para GED (para aplicação futura).

Todos os requisitos que compõem a Certificação SBIS-CFM são apresentados no manual de certificação através de uma tabela, que exibe as seguinte informações:

- **ID**: Identificação do requisito, utilizando codificação padronizada;
- **Requisito**: Nome do requisito;
- **Conformidade**: Descrição do requisito, incluindo exemplos sempre que apropriado. Adicionalmente, pode incluir indicações de como o requisito será avaliado durante a auditoria;
- **Presença**: Esta coluna é sinalizada com os valores M, R ou X. Cada uma destas letras significa respectivamente Mandatório (o requisito deve ser obrigatoriamente atendido pelo S-RES), Recomendado (o requisito é importante, porém ainda não é obrigatório, e possui alta probabilidade de tornar-se obrigatório nas próximas versões do manual) e Não se aplica (o requisito não aplicável à situação apresentada).

Nos requisitos do Nível de Garantia de Segurança 1 (NGS1), a coluna “Presença” está dividida entre “Local” e “Remoto”, refletindo como cada requisito deve ser considerado de acordo com o enquadramento do S-RES que está sendo auditado.

Os requisitos de segurança de um S-RES são fundamentais para garantir a privacidade, confidencialidade e integridade da informação identificada em saúde. Aos interessados em eliminar o registro das informações em papel, é obrigatória a conformidade

ao Nível de Garantia de Segurança 2 (NGS2), que contempla obrigatoriamente o uso de certificados digitais.

O Processo de Certificação SBIS-CFM classifica os S-RES, do ponto de vista de segurança da informação, em dois Níveis de Garantia de Segurança (NGS) [12]:

- **NGS1**: categoria aplicável a S-RES que não pretendem eliminar a impressão dos registros em papel. Assim, mantém a necessidade de impressão e aposição manuscrita da assinatura;
- **NGS2**: categoria constituída por S-RES que viabilizam a eliminação do papel nos processos de registros de saúde. Para isso, especifica a utilização de certificados digitais ICP-Brasil para os processos de assinatura e autenticação. Para atingir o NGS2 é necessário que o S-RES atenda aos requisitos já descritos para o NGS1 e apresente ainda total conformidade com os requisitos especificados para o Nível de Garantia 2.

Para efeito da certificação SBIS-CFM, os S-RES foram classificados em:

- **Acesso Local**: todo S-RES instalado num único computador, com acesso ao sistema apenas neste equipamento. Além disso, um S-RES de acesso local não deverá permitir o acesso simultâneo por mais de um usuário.
- **Acesso Remoto**: todo S-RES que permite o acesso simultâneo ao sistema, no computador onde o S-RES está instalado, ou em computador remoto, através de algum tipo de conexão (rede local, conexão sem-fio, internet, etc.).

Segundo a classificação definida no manual de certificação, o SIAP se enquadra na categoria Básica, no NGS2 e é um sistema de acesso remoto. Para se submeter ao processo de certificação, deve responder pelo menos a todos os requisitos obrigatórios destas categorias.

4.3 Características do SIAP como aplicativo Java EE

Em relação a arquitetura Java EE, o sistema SIAP é formado por componentes distribuídos nas camadas cliente, web e de negócio.

A camada cliente possui os seguintes tipos de componentes:

- Há um módulo de aplicação cliente, que é usado para transmitir os dados capturados em localidades remotas pelas equipes de saúde através de dispositivo móvel para o banco de dados do sistema.

- Páginas geradas pelo módulo de aplicação web, que é por onde as equipes médicas especializadas localizadas no hospital acessam os dados dos pacientes atendidos pelas equipes em campo, dentre outras funcionalidades.

- Um Applet, embutido na página de emissão de laudo, usado para que o médico possa assinar digitalmente um laudo. Nesta situação o Applet se fez necessário porque o processo de assinatura digital envolve o uso de chave privada e esta só deve ser acessada localmente, não pode ser transmitida pela rede.

A camada web foi desenvolvida com a tecnologia JSF e com o auxílio de componentes gráficos da tecnologia RichFaces, um framework de componentes gráficos avançados desenvolvido e distribuído pela organização JBoss sob licença LGPL [10]. Componentes JavaBeans, especificamente Managed Beans (beans gerenciados), também foram utilizados com a finalidade de gerenciar o fluxo de dados entre a camada web e a camada de negócio. Beans gerenciados podem conter referência(s) a objetos da camada de negócio e implementar funcionalidades de validação de dados. A página de cadastro de usuarios do sistema, apresentada na figura 4.1, exemplifica uma interface web do sistema desenvolvida com as tecnologias web listadas.

Em relação a camada de negócio, o sistema é composto especificamente por beans de sessão e beans de entidade, além de usar as facilidades de gerenciamento de transação e de segurança do servidor Java EE. No que diz respeito a camada EIS, o SIAP usa o sistema de gerenciamento de banco de dados MySQL.

A central de atendimento web utiliza os serviços de segurança da plataforma, assim o SIAP implementou autenticação e autorização através da técnica declarativa, com adição de metadados nos arquivos descritores XML em cada módulo de aplicação que compõe o sistema. Dentre as estratégias de autenticação disponibilizadas pela plataforma, a central de atendimento optou pela autenticação baseada em formulário, com proteção da troca de dados entre cliente e servidor através do protocolo HTTPS. A comunicação segura via HTTPS também foi configurada de modo declarativo. Através deste protocolo o servidor troca dados com integridade e confidencialidade com o cliente e também é submetido a autenticação, processo que precede até a autenticação do usuário.

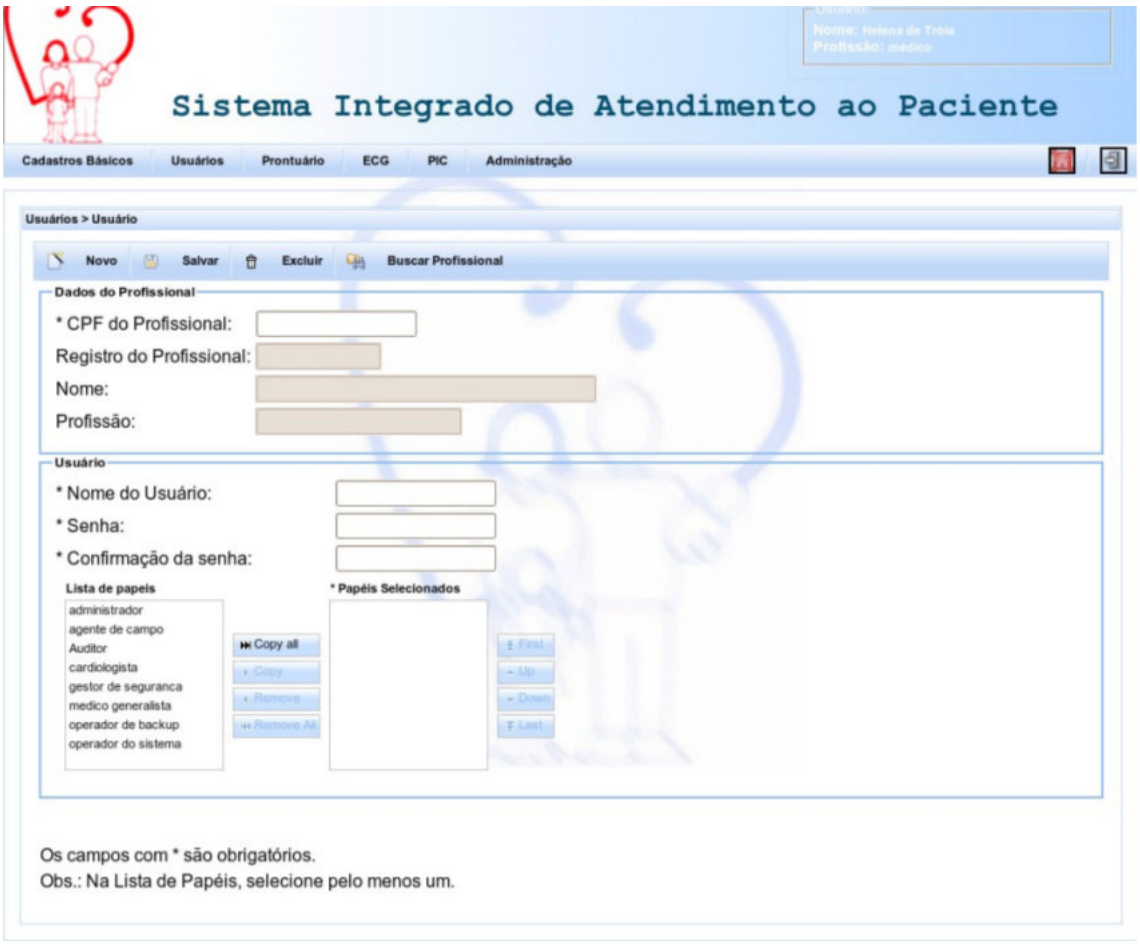

Figura 4.1: página de cadastro de usuário do sistema.

A autenticação do servidor ocorre quando o usuário solicita pela primeira vez o acesso, ocasião em que o servidor apresenta seu certificado digital ao cliente. Caso o usuário confie no certificado apresentado, o protocolo segue com o estabelecimento de uma chave de sessão usada para criptografar os dados trocados entre cliente e servidor durante toda a sessão. Se o usuário aceitar o certificado como permanentemente válido, a etapa de verificação da identidade do servidor passa a ser executada automaticamente pelo cliente web toda vez que ele acessar as páginas do sistema enquanto o certificado digital do servidor permanecer válido. Não apenas a página de login tem garantias de integridade e confidencialidade, mas todas as demais páginas do sistema foram configuradas para transmitir dados via canal seguro, além de terem sido configuradas com acesso restrito, onde apenas os usuários pertencentes a determinados grupos podem acessar e realizar operações sobre as páginas da central de atendimento.

Dentre outras características relacionadas ao processo de autenticação está o fato de os dados de todos os usuários serem armazenados no banco de dados do próprio

sistema, as senhas são armazenadas de forma codificada com o algoritmo de função hash SHA-1 e a qualidade delas é verificada no momento em que o usuário faz seu cadastro (deve ser alfanumérica e com no mínimo de 8 caracteres), cada usuário tem um identificador único e apenas aqueles que são autorizados podem acessar o sistema, a sessão do usuário é desativada após trinta minutos de inatividade e os usuários são atribuídos a papeis/grupos de usuário durante o processo de cadastro. A necessidade de cadastrar dados e grupos de usuários, com estas características, dentro do banco da própria aplicação exigiu a criação de um realm customizado, chamado ECGRealm, adicionado e configurado no servidor de aplicação.

No conjunto de APIs de segurança apresentados no capítulo anterior, a Java Certification Path API foi utilizada para validar os certificados dos usuário que assinaram digitalmente laudos médicos e assim validar as assinaturas. APIs da JCA (Java Cryptography Architecture), que é a parte da plataforma Java que reúne um conjunto de APIs que implementam os mais variados serviços e algoritmos de criptografia, tais como assinatura digital, resumo de mensagens (hashes), certificados e validação de certificados, etc, serviram para adicionar a funcionalidade de assinatura dos dados do laudo na máquina cliente. Todo o conteúdo de um laudo é formatado segundo a sintaxe XML para assinatura digital, conhecida como XML Signature, também suportada pela API Java XML Digital Signature [15]. Os certificados digitais utilizados no sistema foram criados através da ferramenta OpenSSL, um conjunto de ferramentas código aberto que implementa os protocolos SSL (nas versões 2 e 3) e TLS (na versão 1), além de várias bibliotecas de criptografia [11].

4.4 Política de Segurança

É relevante para o sistema ter uma especificação clara de quais tipos de usuários terão acesso a ele, isto inclui a declaração de grupos e papéis que irão existir, e quais funcionalidades cada grupo de usuário terá permissão de executar. O manual de certificação SBIS/CFM é uma importante fonte de informação sobre a política de segurança que deve ser verificada em S-RES, por isso parte dos requisitos de segurança definidos no referido manual podem ser encontrados no sistema objeto de estudo deste trabalho.

Durante a definição da política de segurança foi necessário um estudo do con-

junto de papéis de usuários que deveriam constar no sistema e do tipo de operação que cada usuário poderia realizar segundo o papel que estaria desempenhando, o resultado foi a criação de oito papéis de segurança: cardiologista, agente de campo, operador do sistema, administrador do sistema, gestor de segurança, auditor, operador de backup e medico generalista. Alguns destes papéis constam como obrigatórios no manual SBIS/CFM, este é o caso do papel de operador de backup, por exemplo. Outros são específicos do sistema proposto, o de cardiologista, por exemplo. Em relação ao papel de operador de backup, que consta como obrigatório no manual, sua função é meramente figurativa, pois nenhuma funcionalidade foi implementada na central de atendimento para dar suporte as suas atividades. Cada um dos papeis/grupos de usuários poderá executar as seguintes tarefas:

- cardiologista: visualizar prontuários eletrônicos, cadastrar dados de identificação do paciente, exames físicos, eletrocardiogramas, tipos de alterações eletrocardiográficas conhecidas e emitir laudos referentes a eletrocardiogramas de pacientes;
- médico generalista: as mesmas operações que um cardiologista, exceto a emissão de laudos;
- agente de campo: membro das equipes de saúde que trabalha nas cidades em que os pacientes são atendidos. Dentro do sistema poderá registrar os exames físicos, eletrocardiogramas e dados de identificação dos pacientes coletados em campo;
- operador do sistema: este usuário poderá realizar operações básicas na central de atendimento, tais como cadastrar cidades, bairros, profissões e tipos de alterações eletrocardiográficas não catalogados, além dos profissionais envolvidos nos serviços de atendimento em saúde;
- administrador do sistema: visualizar, cadastrar, excluir e atualizar dado de usuários;
- gestor de segurança: cadastrar, atualizar, visualizar e excluir papeis/grupos de usuário, além de visualizar logs de ações no sistema;
- auditor: apenas visualiza os logs de ações no sistema;
- operador de backup: realizar cópias de segurança e restaurá-las.

No modelo de segurança Java EE todo o controle de acesso é gerenciado pelos contêineres dos componentes web e enterprise. A estrutura de segurança do SIAP segue

este modelo, sendo constituída de arquivos descritores de implantação, onde foram declarados os papeis de usuários existentes para o sistema, regras de controle de acesso e autenticação. Arquivos descritores XML não fazem parte dos arquivos de classes, nem dos arquivos das páginas web da aplicação, eles são lidos pelos contêineres no momento da implantação dos módulos da aplicação dentro do servidor, permitindo aos contêineres gerenciar a aplicação de acordo com suas necessidades de segurança.

Expressar os requisitos de segurança através de arquivos descritores é uma das alternativas da técnica de segurança declarativa, que também oferece a opção de incluí-los por meio da adição de anotações (metadados) diretamente nos arquivos de classe dos componentes, como já explicado no capítulo anterior. No sistema SIAP optou-se pela técnica declarativa porque a solução disponibilizada pela plataforma contempla a maioria das necessidades do sistema. Além disso a inclusão dos requisitos nos arquivos descritores foi preferida em relação ao uso de anotações porque nem todos os metadados podem ser expressos em forma de anotações. Outra vantagem dos arquivos descritores é que as informações de configuração ficam centralizadas em poucos arquivos ao invés de se espalharem por várias classes. Alterar o descritor requer apenas reimplantar a aplicação para que o servidor reconheça as alterações, enquanto alterar anotações implica em mudanças no código fonte e consequentemente reconstrução e reimplantação dos módulos executáveis.

4.5 Autenticação

O processo de autenticação do sistema foi implementado de forma declarativa por meio da inclusão de metadados nos arquivos descritores de implantação *web.xml* e *sun-web.xml* (nas versões mais recentes do servidor de aplicação GlassFish este arquivo foi renomeado para *glassfish-web.xml*). O *web.xml* do projeto foi editado para indicar ao contêiner que a aplicação fará uso da autenticação baseada em formulário e que o controle do processo será feito pelo domínio customizado ECGRealm, que foi registrado no servidor com o nome *ecg*, conforme o trecho do código fonte abaixo:

```
1 <login-config>
2     <auth-method>FORM</auth-method>
3     <realm-name>ecg</realm-name>
4     <form-login-config>
```

```
5          <form-login-page>/loginJSP/login.jsf</form-login-page>
6          <form-error-page>/loginJSP/loginErro.jsf</form-error-page>
7      </form-login-config>
8  </login-config>
```

A figura 3.7, apresentada no capítulo anterior, ilustra o processo de autenticação adotado. Na etapa 3 do processo representado, o formulário de autenticação submetido executa a ação *j_security_check*. Esta ação é implementada pelo próprio servidor de aplicação, que delega ao domínio (realm) *ecg*, apresentado no trecho de código anterior, o processo de autenticação e autorização.

A autenticação via formulário foi a estratégia escolhida pelo SIAP por permitir criar páginas customizadas de login e de erro de login. A página de login deve ter no mínimo o seguintes elementos:

- Um campo de entrada de dados do tipo *"text"* tendo, necessariamente, o atributo *"name"* com o valor *j_username*;
- Um campo de entrada de dados do tipo *"password"* tendo, necessariamente, o atributo *"name"* com o valor *j_password*;
- Um formulário para agrupar os elementos citados tendo, obrigatoriamente, *j_security_check* como valor para o atributo *"action"* e método de envio via POST.

O trecho de código abaixo mostra a organização destes elementos na página de login:

```
1 <form method="POST" action="j_security_check">
2   <input type="text" name="j_username">
3   <input type="password" name="j_password">
4 </form>
```

Quando o formulário é submetido, os valores dos campos são transmitidos com confidencialidade e integridade, pois o módulo web do sistema foi configurado com os seguintes metadados no elemento *security-constraint*:

```
1 <security-constraint>
2     ...
3     <user-data-constraint>
4         <description/>
```

```
5          <transport-guarantee>CONFIDENTIAL</transport-guarantee>
6      </user-data-constraint>
7  </security-constraint>
```

O elemento *security-constraint* do arquivo descritor de implantação *web.xml* do sistema, mostrado no pedaço de código acima, serve para adicionar restrições de segurança a um conjunto de recursos que precisa ser protegido. Ao configurar a forma de transporte de dados como **confidencial**, tanto a integridade quanto a confidencialidade estão garantidas e a comunicação segura via protocolo HTTPS é ativada para proteger a aplicação. O sub-elemento *transport-guarantee* pode assumir os valores CONFIDENTIAL, INTEGRAL ou NONE. Na prática, não há diferença no funcionamento quando um dos dois primeiros valores é escolhido, já o último indica dispensa total da comunicação segura. Além do sub-elemento *user-data-constraint*, que restringe a visualização e a alteração dos dados durante o envio, outros elementos também podem ser aninhados dentro do *security-constraint*, estes serão detalhados mais a frente quando a implementação do controle de acesso do sistema for abordada.

O protocolo HTTPS requer que o servidor tenha um par de chaves privada e pública, além do certificado digital correspondente. A figura 4.2 representa o processo de autenticação mútua, onde o cliente é autenticado através das credenciais de nome de usuário e senha, enquanto o servidor apresenta seu certificado. As etapas ilustradas são as seguintes [3]:

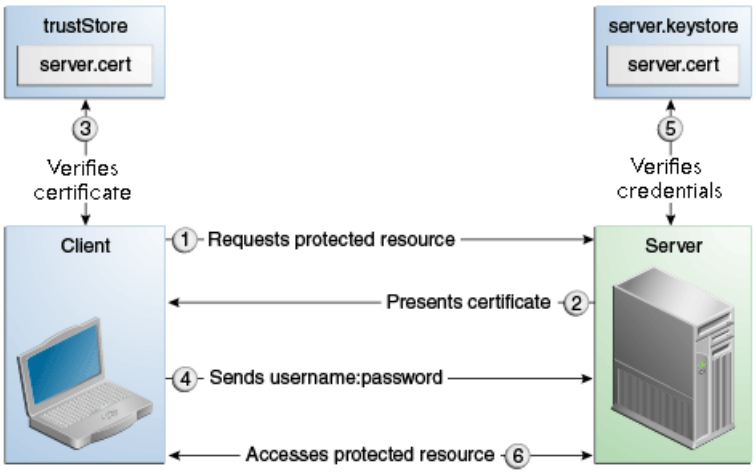

Figura 4.2: autenticação mútua baseada em credenciais [3].

1. O cliente solicita acesso a um recurso protegido.

2. O servidor web apresenta seu certificado ao cliente.

3. O cliente possui um banco de dados de certificados confiáveis (*trustStore*), que pode estar instalado no próprio navegador, por exemplo, e verifica se o certificado do servidor faz parte ou está relacionado a este conjunto confiável.

4. Se o certificado for válido o cliente envia seu nome de usuário e senha para o servidor criptografados com a chave pública do mesmo.

5. O servidor verifica as credenciais do cliente, para isso precisa de sua própria chave privada, protegida no *keystore* junto com seu certificado digital.

6. Se a verificação ocorrer com sucesso, o servidor concede acesso ao recurso protegido solicitado pelo cliente.

No momento da instalação do servidor GlassFish um par de chaves pública e privada, junto com um certificado digital auto-assinado são criados. Este certificado auto-assinado é usado por padrão e é apresentado como certificado do servidor durante a execução do protocolo HTTPS. Para ser implantada em um ambiente de produção, a central de atendimento precisaria usar um certificado assinado por uma autoridade certificadora homologada pela Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil), conforme exige o manual de certificação SBIS/CFM [12].

Quando o controle de acesso é adicionado aos componentes da camada enterprise, a autenticação é automaticamente exigida pelo contêiner EJB. Se nenhuma estratégia de autenticação é definida nas camadas que consomem EJBs e os métodos dos beans são demarcados como métodos protegidos, o contêiner aplica a autenticação básica, lançando ao cliente uma janela de diálogo para autenticação com nome de usuário e senha [3].

4.6 Controle de Acesso

O sistema SIAP optou pelo mecanismo de autorização fornecido pelos serviços de segurança da plataforma Java EE. O controle de acesso foi incluído de forma declarativa, através da adição de metadados nos arquivos descritores de implantação dos módulos do projeto. Sempre que ocorre a solicitação de acesso a uma página web ou método de

EJB protegido, o contêiner repassa ao realm configurado a verificação dos direitos de acesso.

A figura 4.3 representa o processo de controle de acesso. Na representação, o contexto de segurança guarda informações (credenciais) das entidades que passaram por autenticação. As credenciais das entidades são propagadas do contexto de segurança do contêiner web para o contexto do contêiner EJB [3]. Quando requisições a recursos protegidos chegam, o contêiner identifica quais papeis foram registrados na aplicação como autorizados a operar sobre o recurso solicitado. Tendo identificado os papeis, o contêiner consulta as regras de mapeamento entre papeis e grupos ou principals da aplicação, assim, sabe exatamente os grupos e/ou principals autorizados. O passo seguinte é verificar se a entidade que faz uma requisição, identificada pelas credenciais, pertence a algum dos grupos ou principals levantados, tarefa que é delegada ao realm. Ao final, o contêiner terá como resultado a concessão de direitos de acesso, ou lançará uma exceção, caso a entidade não pertença ao grupo ou não possua o principal mapeado.

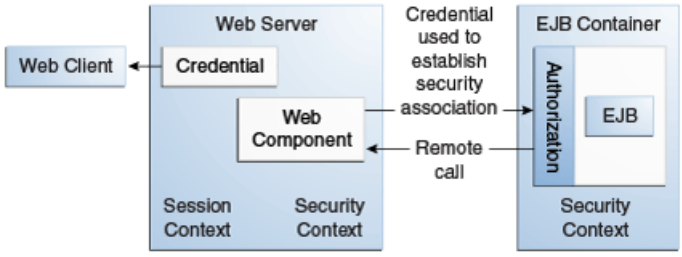

Figura 4.3: invocação de método em um EJB e autorização [3].

O módulo web da central de atendimento restringe o acesso as páginas do sistema apenas aos usuários que possuem um dos oito papeis definidos na política de segurança.

```
1  <security-constraint>
2      <display-name>RetricoesDeAcessoWeb</display-name>
3      <web-resource-collection>
4          <web-resource-name>ColecaoDeRecursosWeb</web-resource-name>
5          <url-pattern>/jsp/*</url-pattern>
6          <http-method>GET</http-method>
7          <http-method>POST</http-method>
8      </web-resource-collection>
9      <auth-constraint>
10         <role-name>cardiologista</role-name>
11         <role-name>operador_do_sistema</role-name>
```

```
12          <role-name>agente_de_campo</role-name>
13          <role-name>auditor</role-name>
14          <role-name>operador_de_backup</role-name>
15          <role-name>administrador</role-name>
16          <role-name>gestor_de_seguranca</role-name>
17          <role-name>medico_generalista</role-name>
18      </auth-constraint>
19      <user-data-constraint>
20          <transport-guarantee>CONFIDENTIAL</transport-guarantee>
21      </user-data-constraint>
22  </security-constraint>
```

No código acima, pertencente ao arquivo *web.xml*, todas as requisições via GET ou POST as páginas da aplicação web cujo caminho coincida com /jsp/* só podem ser acessadas por usuários que estejam relacionados a pelo menos um dos papeis listados no elemento *auth-constraint*. Na prática, todas as páginas do sistema estão com acesso restrito, pois todas elas estão localizadas dentro do sub-diretório jsp.

Para que as restrições de acesso as páginas tenham efeito é necessário que todo o módulo web conheça os papeis existentes, por isso, no mesmo arquivo deve haver a declaração dos oito papeis existentes para a aplicação:

```
1  <security-role>
2      <role-name>cardiologista</role-name>
3  </security-role>
4  <security-role>
5      <role-name>operador_do_sistema</role-name>
6  </security-role>
7      ...
8  <security-role>
9      <role-name>medico_generalista</role-name>
10 </security-role>
```

As declarações são elementos *security-role*, que podem ser escritos em qualquer parte do *web.xml*, desde que estejam aninhados diretamente ao elemento raiz do arquivo. O mapeamento que relaciona os papeis da aplicação com os grupos existentes no realm ecg é registrado no arquivo *sun-web.xml*:

```
1  <security-role-mapping>
2    <role-name>cardiologista</role-name>
3    <group-name>cardiologista</group-name>
```

```
</security-role-mapping>
<security-role-mapping>
  <role-name>agente_de_campo</role-name>
  <group-name>agente_de_campo</group-name>
</security-role-mapping>
  ...
<security-role-mapping>
  <role-name>medico_generalista</role-name>
  <group-name>medico_generalista</group-name>
</security-role-mapping>
```

No trecho acima os nomes de papeis e grupos coincidem, no entanto isto não é obrigatório. Grupos de usuário representam algo mais abrangente, perfis de usuário registrados no servidor que podem ser utilizados por várias aplicações, enquanto papeis são específicos para uma aplicação.

O módulo enterprise da aplicação também registra o mapeamento entre grupos e papeis. Este mapeamento usa os mesmos elementos e valores de metadados do módulo web, no entanto, ficam registrados no arquivo *sun-ejb-jar.xml*. O controle de acesso na camada de negócio foi aplicado através da adição metadados de segurança no arquivo *ejb-jar.xml*. Para cada papel foi registrado o conjunto de métodos sobre o qual tem permissão de acesso.

```
<assembly-descriptor>
    <security-role>
        <role-name>cardiologista</role-name>
    </security-role>
    <security-role>
        <role-name>operador_do_sistema</role-name>
    </security-role>
        ...
    <security-role>
        <role-name>medico_generalista</role-name>
    </security-role>

    <method-permission>
        <role-name>cardiologista</role-name>
        <method>
            <ejb-name>GerenciadorDocumentosCardiologiaBean</ejb-name>
            <method-name>inserirAlteracaoEletrocardiografica</method-name>
```

```
18          </method>
19          <method>
20              <ejb-name>GerenciadorDocumentosCardiologiaBean</ejb-name>
21              <method-name>alterarAlteracaoEletrocardiografica</method-name>
22          </method>
23          ...
24          <method>
25              <ejb-name>GerenciadorDocumentosCardiologiaBean</ejb-name>
26              <method-name>listarEcgPendentesPorMedico</method-name>
27          </method>
28      </method-permission>
29
30      ...
31
32      <method-permission>
33          <role-name>administrador</role-name>
34          <method>
35              <ejb-name>GerenciadorSegurancaBean</ejb-name>
36              <method-name>listarTodosUsuarios</method-name>
37          </method>
38          <method>
39              <ejb-name>GerenciadorSegurancaBean</ejb-name>
40              <method-name>inserirUsuario</method-name>
41          </method>
42          ...
43          <method>
44              <ejb-name>GerenciadorSegurancaBean</ejb-name>
45              <method-name>consultarUsuarioPorCPF</method-name>
46          </method>
47      </method-permission>
48      <exclude-list>
49          <method>
50              <ejb-name>GerenciadorProntuarioBean</ejb-name>
51              <method-name>excluirPaciente</method-name>
52          </method>
53      </exclude-list>
54  </assembly-descriptor>
```

O código acima apresenta o elemento *assembly-descriptor*, que pode ser adicionado em qualquer parte do arquivo *ejb-jar.xml*, desde que seja aninhado ao elemento

raiz do arquivo. O *assembly-descriptor* reúne a declaração de cada um dos oito papeis existentes no sistema, bem no início do código, e o conjunto de métodos permitidos a cada papel, logo abaixo da declaração de papeis. O elemento *exclude-list* serve para indicar o conjunto de métodos que não pode ser executado por nenhum papel. Estas regras de controle de acesso derivam da política de segurança definida para o sistema.

No mesmo arquivo descritor, o *ejb-jar.xml*, todos os EJBs da aplicação devem ser declarados, pois o contêiner EJB precisa reconhecer os beans existentes antes de aplicar as restrições de acesso descritas no *assembly-descriptor*. As classes de componentes EJBs podem até conter anotações que informem ao contêiner que se tratam de beans de sessão, no entanto, o conteúdo do arquivo descritor de implantação tem prioridade e sobrepõe qualquer tipo de anotação correspondente. Além do mais, o servidor pode não carregar informações decorrentes das anotações das classes no contexto do contêiner quando parte delas se encontra nos descritores de implantação, assim, para que as regras de acesso do *ejb-jar.xml* funcionem corretamente é melhor incluir as declarações de EJBs dentro deste arquivo, como no código a seguir:

```
1  <enterprise-beans>
2    <session>
3      <ejb-name>GerenciadorLogradouroBean</ejb-name>
4      <remote>br.ufma.lsd.siap.ejb.base.GerenciadorLogradouroRemote</remote>
5      <ejb-class>br.ufma.lsd.siap.ejb.base.GerenciadorLogradouroBean</ejb-class>
6      <session-type>Stateless</session-type>
7    </session>
8    <session>
9      <ejb-name>GerenciadorProntuarioBean</ejb-name>
10     <remote>br.ufma.lsd.siap.ejb.prontuario.GerenciadorProntuarioRemote</remote>
11     <ejb-class>
12       br.ufma.lsd.siap.ejb.prontuario.GerenciadorProntuarioBean
13     </ejb-class>
14     <session-type>Stateless</session-type>
15   </session>
16
17   ...
18
19   <session>
20     <ejb-name>GerenciadorPICBean</ejb-name>
21     <remote>br.ufma.lsd.siap.ejb.pic.GerenciadorPICRemote</remote>
22     <ejb-class>br.ufma.lsd.siap.ejb.pic.GerenciadorPICBean</ejb-class>
```

```
23      <session-type>Stateless</session-type>
24    </session>
25  </enterprise-beans>
```

O elemento *enterprise-beans* reúne a declaração de todos os EJBs do sistema e pode ser adicionado em qualquer parte do *ejb-jar.xml*, desde que esteja aninhado diretamente ao elemento raiz do arquivo.

As regras de permissão aplicadas através da demarcação dos métodos dos beans e o mapeamento entre papeis e grupos encerram parte da implementação da política de segurança, o restante da política do sistema é concluída com a implementação de um realm customizado. Os arquivos das classe ECGLoginModule e ECGRealm, junto a outros de classes utilitárias, foram empacotados dentro de um arquivo JAR e adicionados dentro do servidor GlassFish, especificamente no diretório de bibliotecas do servidor (*glassfish4/glassfish/lib*, por exemplo). A classe ECGLoginModule está indiretamente relacionada a classe LoginModule do framework JAAS [16]. Classes que implementam a interface LoginModule tem como principal função submeter ao processo de autenticação a entidade que solicita acesso a um recurso sensível e atribuir a ela informações de identidade, credenciais e até identidade de grupo quando a entidade consegue comprovar sua identidade.

A criação de um realm customizado exige a escrita de uma classe que estenda **com.sun.appserv.security.AppservPasswordLoginModule** e implemente o método abstrato *authenticateUser* herdado. A sub-classe escrita não pode implementar a interface LoginModule diretamente [17]. ECGLoginModule pode acessar os atributos *_username* e *_password* de AppservPasswordLoginModule, que é onde a superclasse armazena os valores das credenciais capturadas durante a interação com o usuário. O atributo *_currentRealm* da superclasse serve para que ECGLoginModule tenha acesso ao realm e possa verificar através dele se as credenciais apresentadas são válidas. Além disso, através do atributo *_currentRealm*, ECGLoginModule pode obter informações sobre os usuários cadastrados no domínio, tal como os grupos a que pertencem. ECGLoginModule deve finalizar o método *authenticateUser* com uma chamada ao método *commitUserAuthentication*, atribuindo os grupos recuperados do realm ao usuário que acabou de ser autenticado.

```
1 public class ECGLoginModule extends AppservPasswordLoginModule {
2
3 protected void authenticateUser() throws LoginException {
```

```
4       if (!(this._currentRealm instanceof ECGRealm)) {
5           throw new LoginException("...");
6       }
7       ECGRealm ecgRealm = (ECGRealm)this._currentRealm;
8       String[] roles =
9         ecgRealm.authenticateUser(this._username, this._password);
10      if (roles == null) {
11          throw new LoginException("...");
12      }
13      this._subject.getPrincipals()
14        .add(new ECGPrincipal(this._username));
15      commitUserAuthentication(roles);
16  }
17  }
```

Quando o sistema SIAP foi desenvolvido não havia um bom discernimento sobre a diferença entre papel e grupo. Na linha 8 do código apresentado logo acima, o atributo roles deveria se chamar groups, pois o realm lida apenas com grupos e principals de usuários, papel é um conceito do escopo da aplicação, não do domínio.

Além de implementar um AppservPasswordLoginModule, um custom realm deve possuir também uma extensão de **com.sun.appserv.security.AppservRealm** [17]. A sub-classe estendida será a ponte de acesso ao banco de dados de usuários, grupos, credenciais e principals armazenados em um espaço de domínio. Quando o servidor de aplicação é iniciado, o método *init* de um realm é o primeiro a ser chamado. A classe ECGRealm obtém através deste método propriedades de conexão com o banco de dados do sistema. Estas propriedades são cadastradas no servidor de aplicação pelo administrador do servidor, no momento em que registra o realm customizado **ecg**, conforme ilustra a figura 4.4.

Durante o cadastro do domínio ecg no servidor GlassFish as seguintes propriedades foram definidas:

- **jaas-context**: é o nome que corresponde a uma das entradas do arquivo de configuração de login do servidor de aplicação. No servidor GlassFish, versão 4, este arquivo se localiza em *glassfish4/glassfish/domains/domain1/config/login.conf*. Cada entrada deste arquivo indica ao framework JAAS qual conjunto de classes LoginModule serão usadas em um determinado processo de autenticação. O valor deste

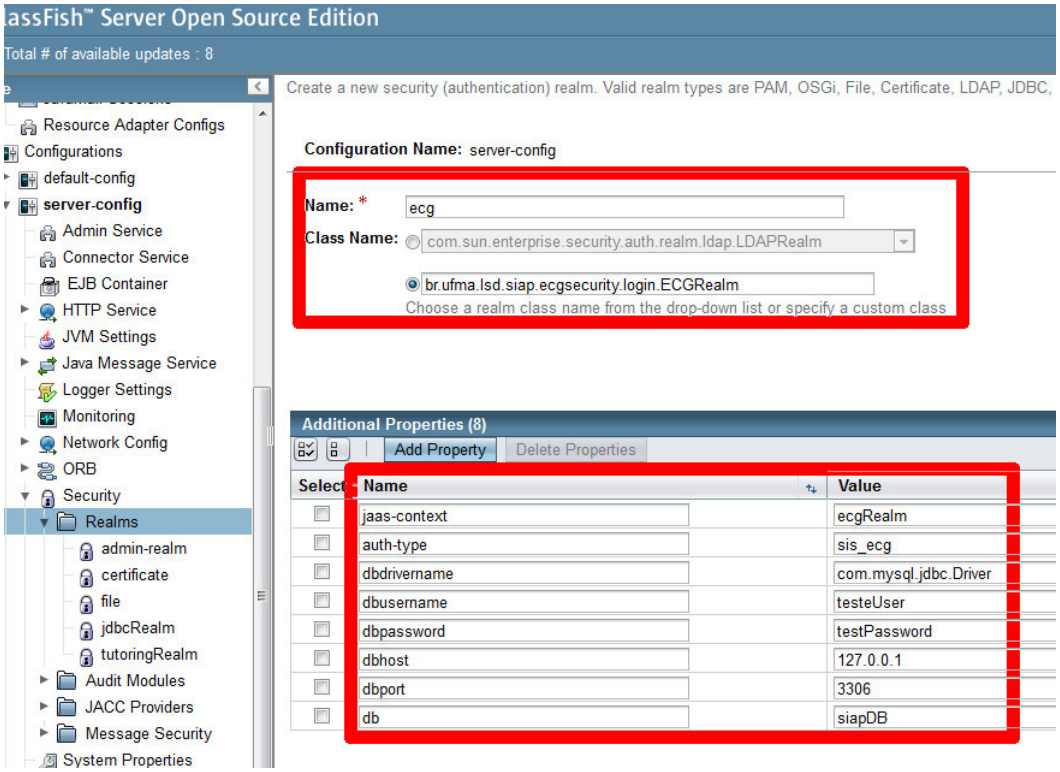

Figura 4.4: configuração do realm ecg no servidor de aplicação através da ferramenta Admin Console do servidor GlassFish.

atributo em nosso sistema é *ecgRealm* e foi registrado no arquivo de configuração conforme o código abaixo.

```
1 ecgRealm {
2     br.ufma.lsd.siap.ecgsecurity.login.ECGLoginModule required;
3 };
```

Esta entrada indica qual classe realizará a autenticação. Se houvesse necessidade de efetuar autenticação através de vários sistemas, bastava adicionar mais linhas na mesma entrada, indicando as classes responsáveis pela autenticação. O gerenciamento de transação controla a execução da pilha de autenticações em uma entrada. Ao lado da declaração de cada classe deverá haver um flag de controle de "aceitabilidade de sucesso", indicando se a autenticação efetuada pela classe é mandatória ou opcional. O flag indica como o sucesso ou falha de um LoginModule afeta o

sucesso total de uma tentativa de autenticação e pode assumir um dos seguintes valores: required, requisite, sufficient ou optional [13]. Aplicar vários métodos de autenticação de uma vez é útil para agregar num mesmo sujeito todas as identidades que permitem identificá-lo em diferentes sistemas integrados e facilitar verificações de controle de acesso.

O arquivo de configuração de login mantém entradas para outras implementações de LoginModule prontas para serem utilizadas por aplicações Java EE.

```
1 ldapRealm {
2     com.sun.enterprise.security.auth.login.LDAPLoginModule required;
3 };
4 ...
5 pamRealm {
6     com.sun.enterprise.security.ee.auth.login.PamLoginModule required;
7 };
```

- **auth-type**: é apenas um nome ou uma descrição que representa o tipo de autenticação feita pelo realm. Em nosso sistema apelidamos a autenticação de *sis_ecg*, esta informação é exibida no console do servidor de aplicação quando ele é iniciado, indicando que o realm *sis_ecg* também foi iniciado.
- **db**: o nome do banco de dados que armazena dados dos usuários do sistema.
- **dbusername**: informa o nome do usuário do banco de dados.
- **dbpassword**: indica a senha do usuário do banco de dados.
- **dbdrivername**: indica qual API será usada para conexão e interação com um tipo específico de gerenciador de banco de dados. No sistema SIAP o valor atribuído a esta propriedade é *com.mysql.jdbc.Driver*.
- **dbhost**: informa o endereço de localização do servidor onde o banco de dados está instalado. Se o servidor de banco de dados estiver localizado na mesma máquina em que está instalado o servidor de aplicação, então esta propriedade pode ser cadastrada com o valor localhost.
- **dbport**: o número da porta do servidor de banco de dados.

A classe ECGLoginModule delega a ECGRealm o processo de autenticação, que verifica as credenciais do usuário consultando o banco de dados. Os grupos de usuário

também são recuperados através de consulta ao banco de dados. As propriedades de conexão com o banco de dados são obtidas na inicialização do domínio e seus valores são salvos nos atributos da classe, veja o trecho de código abaixo.

```
1  public class ECGRealm extends AppservRealm {
2      ...
3      private String authType;
4      private String dbDriverName;
5      ...
6      @Override
7      public void init(Properties props) throws BadRealmException,
8                NoSuchRealmException {
9          super.init(props);
10         ...
11         String jaasCtx = props.getProperty("jaas-context");
12         if(jaasCtx == null)
13             throw new BadRealmException("...");
14         setProperty("jaas-context", jaasCtx);
15         this.dbhost = props.getProperty(DB_HOST);
16         ...
17     }
18     @Override
19     public Enumeration getGroupNames(String user)
20             throws InvalidOperationException,
21                 NoSuchUserException {
22               ...
23     }
24     ...
25     public String[] authenticateUser(String user,
26             String password) {
27         ...
28     }
29     ...
30 }
```

4.7 Auditoria

Quanto ao mecanismo de auditoria, fez-se uso da técnica programática através da obtenção de informações do contexto de segurança do contêiner, tal como o nome do

usuário registrado no sistema:

```
1  @Resource SessionContext ctx;
2  ...
3  Principal caller = ctx.getCallerPrincipal();
4  codUsuario = caller.getName();
```

As informações obtidas sobre o usuário são registradas no banco de dados juntamente com o tipo de operação que ele está tentando executar: o nome do método acessado e os parâmetros passados, por exemplo. O bean de sessão GerenciadorAuditoriaBean é responsável por cadastrar as operações realizadas:

```
1  package br.ufma.lsd.siap.ejb.seguranca;
2  ...
3  @Stateless
4  public class GerenciadorAuditoriaBean implements
5          GerenciadorAuditoriaRemote {
6      public void registrarLogExcecao(String codusuario,
7          String nomeClasse,
8          String nomeMetodo,
9          String dadosEntrada,
10         String dadosSaida) {
11         ...
12     }
13     public void registrarLogOperacao(String codusuario,
14         String nomeClasse,
15         String nomeMetodo,
16         String dadosEntrada,
17         String dadosSaida) {
18         ...
19     }
20 }
```

O registro da operação efetuada pelo usuário nas trilhas de auditoria do sistema está presente em praticamente todos os métodos das classes de beans do módulo enterprise. Os únicos métodos que não precisam incluir chamadas aos métodos do bean GerenciadorAuditoriaBean são aqueles que apenas listam ou realizam consultas sobre dados de cadastros básicos, tais como listar cidade, bairro ou unidade de saúde. Qualquer operação realizada sobre dados de informação de saúde dos pacientes deve ser registrada no log de operações. O registro de dados nas trilhas de auditoria é seletivo porque adici-

onam mais custo computacional as operações de um bean.

A tabela tbLogOperacoes armazena informações sobre as operações realizadas no sistema e contêm os seguintes atributos:

- idOperacao, um número inteiro sequencial que identifica de maneira única cada registro de auditoria;
- codUsuario, o atributo de identidade do sujeito (código do usuário ou nome de usuário) que está realizando a operação;
- nomeClasse, nome da classe que possui o método que corresponde a operação que está sendo realizada pelo usuário;
- nomeMetodo, nome do método que corresponde a operação que está sendo realizada pelo usuário;
- TSOperacao, a data e hora no formato "ano-mês-dia hora:minuto:segundo", ou carimbo de tempo (timestamp), que registra o momento em que o usuário realizou a operação;
- dadosEntrada, os valores dos parâmetros passados ao método chamado. Os parâmetros são concatenados em uma String separados por ";", como tokens;
- dadosSaida, valor retornado pelo método chamado;

Como os métodos dos beans são todos protegidos com acesso restrito, todo sujeito que requisitar uma operação terá passado no mínimo pelo processo de autenticação. Mesmo se houvesse alguma falha de segurança, o contêiner EJB associaria ao sujeito que acessa o sistema a identidade de ANONYMOUS, que é a identidade associada quando a autenticação não é requerida. Assim, se uma requisição for feita por um sujeito anônimo, o mecanismo de auditoria implementado registraria a operação como um log de exceção ou falha de segurança.

4.8 Assinatura Digital

Dentro do sistema SIAP, a assinatura digital foi uma das operações exigidas pela funcionalidade de emissão de laudo presente na aplicação Web. Após o médico

analisar informações sobre o sinal do ECG, em conjunto com outras informações de saúde do paciente através do sistema, deverá assinar o laudo correspondente usando sua chave privada. A tela de assinatura consiste de um applet que permite a realização de todas as operações criptográficas envolvendo a chave privada do lado cliente.

Para o applet funcionar corretamente do lado cliente é necessário assinar digitalmente o arquivo JAR que o contém. Quando o applet é apresentado do lado cliente, o navegador avisa que o applet foi assinado digitalmente e apresenta o certificado de chave pública de quem o assinou. Se o usuário confiar no certificado apresentado, poderá usar a chave pública presente nele para descriptografar a assinatura gerada com a chave privada correspondente e assim constatar a autenticidade do JAR. Uma vez que o applet tenha sido considerado confiável, terá acesso ao sistema de arquivo local e, consequentemente, a chave privada do médico que irá assinar o laudo. A ferramenta jarsigner, que faz parte do ambiente de desenvolvimento Java, foi utilizada para gerar o applet assinado através do seguinte comando:

```
1  jarsigner -keystore <caminho-01>/jaciara.p12
2  -storetype pkcs12 -signedjar <caminho-02>/ECG-assinatura.jar
3  <caminho-03>/ECG-assinatura.jar jaciara
```

Todo o comando acima é digitado em uma única linha. <caminho-01> representa a localização do arquivo (keystore) que armazena a chave privada, a chave pública e o certificado de chave pública correspondente. <caminho-02> é a localização do arquivo JAR a ser assinado e <caminho-03> é o local de destino do JAR assinado. O keystore é um arquivo que armazena objetos criptográficos, em nosso sistema ele foi produzido com a ferramenta OpenSSL no formato PKCS12, que protege o acesso a chave privada com uma senha definida pelo proprietário e armazena certificados no padrão X.509. O último parâmetro do comando é um *alias*, que é um nome, ou apelido, dado como entrada para uma chave privada e o certificado de chave pública correspondente, podendo ainda haver uma cadeia de confiança associada. Um arquivo PKCS12 pode ter várias entradas, por isso é necessário especificar qual entrada de chave privada será utilizada.

A ferramenta OpenSSL também foi utilizada para simular uma hierarquia de autoridades certificadoras. Assim, foi possível utilizar a criptografia de chave pública de modo semelhante ao uso em ambientes de produção que possuem certificados homologados pela ICP-Brasil verdadeira. Na nossa estrutura hierárquica há uma autoridade certificadora raiz com certificado auto-assinado chamada ICP-Brasil, uma autoridade in-

termediária chamada AC LSD cujo certificado foi emitido pela autoridade raiz e um certificado de usuário final, assinado pela autoridade certificadora intermediária. A figura 4.5 mostra um certificado de usuário.

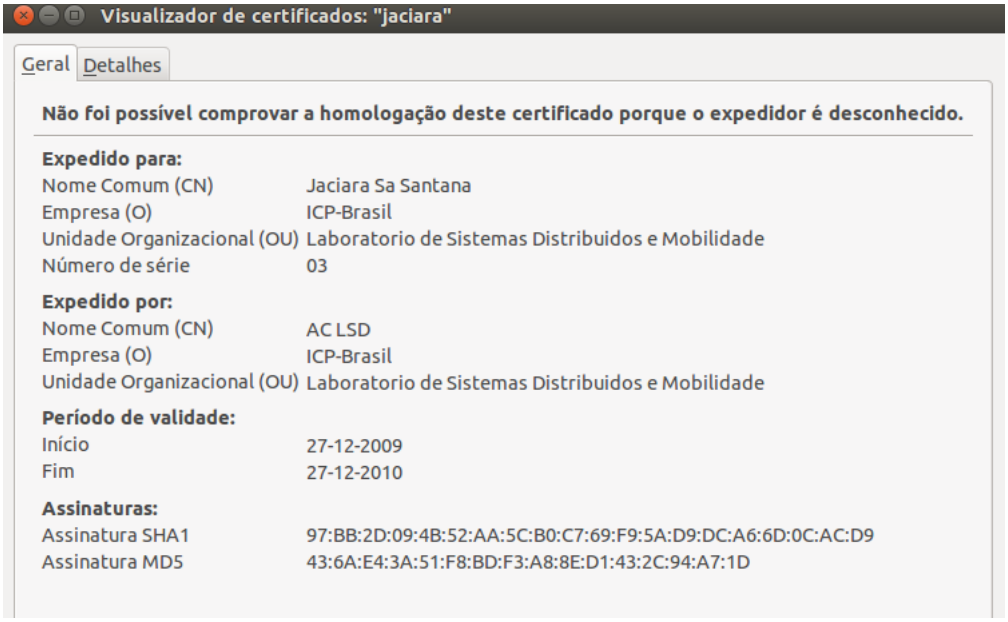

Figura 4.5: certificado de usuário final exibido no visualizador de certificados.

Toda a estrutura formulada serviu para compreender o processo requisição de emissão, emissão e revogação de certificados digitais, emissão e atualização de listas de certificados revogados e desta forma validar o processo de assinatura e verificação de assinatura implementado no sistema.

Apesar de não usar certificados ICP-Brasil verdadeiros, o sistema foi projetado para baixar e atualizar diariamente os certificados das ACs vinculadas a ICP-Brasil e a lista de certificados revogados por cada uma delas. A ICP-Brasil dispunha de um ponto de distribuição destes certificados em seu portal. O banco de dados do sistema armazena os certificados e as listas de certificados nas tabelas *tbCertificadosAC* e *tbLCR*. A tabela *tbCertificadosAC* é formada pelos campos:

1. dnPossuidor;
2. dnEmissor;
3. serialNumber;
4. certificado;
5. nomeArqLCR;

Os três primeiros são dados do tipo varchar e funcionam como a chave primária da tabela. Eles correspondem a atributos presentes no próprio arquivo de cada certificado. O campo certificado é do tipo blob e armazena os bytes do arquivo do certificado digital, enquanto nomeArqLCR é a chave estrangeira que vincula o certificado de uma AC a lista de certificados revogados emitidos por ela, o valor deste atributo deriva de um campo de extensão não obrigatório do certificado que indica o ponto de distribuição da lista de certificados revogados, geralmente um URI.

A tabela *tbLCR* é formada pelos campos nomeArqLCR e lcr. O primeiro é um identificador do tipo varchar que deriva do endereço de localização do ponto de distribuição da lista de certificados revogados, o segundo é um campo do tipo blob que armazena os bytes do arquivo da LCR.

Atualizações diárias nestas tabelas eliminarão os certificados de ACs que forem revogados e irão adicionar certificados renovados ou emitidos para novas autoridades credenciadas. O objetivo de manter certificados e LCRs no banco do sistema é permitir que durante a validação das assinaturas dos laudos o sistema também verifique se o certificado apresentado com a assinatura foi emitido por uma AC considerada confiavel e se este certificado não foi revogado enquanto seu prazo de expiração não foi atingido. O atualizador foi empacotado em um arquivo JAR e projetado para ser executado periodicamente pelo serviço *cron* do servidor linux.

XML Digital Signature foi a tecnologia utilizada para gerar as assinaturas dos laudos, características como facilidade de representar e validar os dados que estão sendo assinados e interoperabilidade foram determinantes para a escolha desta tecnologia. Quando um médico cardiologista cadastra um laudo referente ao exame eletrocardiográfico de um paciente, os dados do formulário submetido são capturados e organizados numa representação própria de metadados:

```
<laudo>
    <perito>
        <cpf>65370350310</cpf>
        <nome>Helena de Sousa</nome>
    </perito>
    <paciente>
        <id>2</id>
        <nome>Antonio Marcos</nome>
        <numero_do_prontuario>2</numero_do_prontuario>
```

```
    </paciente>
    <ecg>
        <id>4</id>
        <data_de_Realizacao data="19/06/2010"/>
        <id_exame_fisico_associado>5</id_exame_fisico_associado>
    </ecg>
    <alteracoes_eletrocardiograficas>
        <alteracao_eletrocardiografica
            codigo_da_alteracao_eletrocardiografica="001"/>
    </alteracoes_eletrocardiograficas>
    <descricao>
        Segmento ST sugestivo de isquemia miocardica.
        O paciente precisa realizar exames mais aprofundados.
    </descricao>
</laudo>
```

O sistema web retorna ao cliente o laudo submetido em forma de metadados através do applet de assinatura. O applet, que é uma aplicação Java padrão, dispõem de um conjunto de APIs criptográficas. O usuário interage com o applet fornecendo a localização do keystore e a senha que dá acesso a chave privada e ao certificado de chave pública para assinar o laudo, resultando numa assinatura digital xml com a seguinte estrutura:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<laudo>
  <perito>
    <cpf>65370350310</cpf>
    <nome>Helena de Sousa</nome>
  </perito>
  <paciente>
    <id>2</id>
    <nome>Antonio Marcos</nome>
    <numero_do_prontuario>2</numero_do_prontuario>
  </paciente>
  <ecg>
    <id>4</id>
    <data_de_Realizacao data="19/06/2010"/>
    <id_exame_fisico_associado>5</id_exame_fisico_associado>
  </ecg>
  <alteracoes_eletrocardiograficas>
```

```
    <alteracao_eletrocardiografica
      codigo_da_alteracao_eletrocardiografica="001"/>
  </alteracoes_eletrocardiograficas>
  <descricao>
    Segmento ST sugestivo de isquemia miocardica.
    O paciente precisa realizar exames mais aprofundados.
  </descricao>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="">
        <Transforms>
        <Transform
          Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signa-ture"/>
        </Transforms>
        <DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>
          jre/9P48jRP7Fakc/YKfO566oAw=
        </DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>
    iVFmZPAEUh6xM3WANNUHW3os...KD3TKsZ2dK26WUyLOU089cv1nr7ge
    LDuvZVF//giaTNenBhvF6iZ4...aSAG8FTpTgwlkCIsxlj0mw8zQoB/V
    1tlc9HIdgXhEoEtq9T4=
    </SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509Certificate>
        MIIEszCCA5ugAwIBAgIBAzA...QYDVQQGEwJCUjETMBEGA1UEChMK
        SUNQLUJyYXNpbDE6MDgGA1U...MxTGFib3JhdG9yaW8gZGUgU2lzd
        IGUgTW9iaWxpZGFkZTEPMA0...UEAxMGQUMgTFNEMB4XDTA5MTIyN
        MjUzMVowgY4xCzAJBgNVBAY...JSMREwDwYDVQQIEwhNYXJhbmhhb
        ...
        LjVDQM/fzlk9R32IAKzLc+c...o/HQIDAQABo4IBvDCCAbgwCQYDV
        BeAwLAYJYIZIAYb4QgENBB8...9wZW5TU0wgR2VuZXJhdGVkIENlc
```

```
        BBR/8sE4BS1eLkwaypAwFmH...r0DzCBwQYDVR0jBIG5MIG2gBRRA
        UhpTcgbW6BdrFj+1MLlIZZd...B6d7YftmNBOWt28vMXH6D8c3b/c
        CTPjZ8OEN2QjJg==
        </X509Certificate>
      </X509Data>
    </KeyInfo>
  </Signature>
</laudo>
```

Esta assinatura digital XML é do tipo *enveloped*, onde a assinatura é aninhada dentro do próprio conteúdo a ser assinado [15]. A raiz do documento é o elemento laudo, os metadados que limitam a assinatura sobre o laudo estão dentro do elemento Signature, que por sua vez está aninhado dentro da raiz laudo. Dentro do elemento Signature há subelementos que informam o algoritmo de resumo de mensagem e de criptografia utilizados, os valores resultantes da aplicação destes algoritmos e a chave pública que deve ser utilizada para verificar a assinatura. O elemento *Reference*, com o atributo URI com valor vazio indica que o conteúdo que consta na assinatura está localizado na raiz do arquivo. Os metadados que caracterizam os métodos empregados na assinatura não são processados junto ao conteúdo assinado.

Apesar de a chave pública ser enviada junto com a assinatura, o processo de validação de assinatura, realizada no servidor, é feita através do certificado de chave pública enviado pelo applet. O certificado tem que ser assinado por uma das autoridades cadastradas no banco de dados do sistema e não pode constar em nenhuma LCR. Só após a validação do certificado é que o processo segue com o uso da chave pública presente no mesmo para analisar a validade da assinatura enviada. Se a assinatura for válida, o laudo é salvo no sistema.

4.9 Conclusão

Este capítulo apresentou o modelo de segurança do sistema SIAP, fazendo uma descrição geral sobre o sistema, suas características como aplicação corporativa Java EE, a política definida para mantê-lo como um sistema seguro, os mecanismos de autenticação e autorização empregados através do reuso dos serviços da plataforma Java EE e a extensão do controle de acesso às necessidades específicas dele. Além disso, também foi abordada a

forma programática como a auditoria foi implementada e como o processo de assinatura digital foi desenvolvido.

5 Conclusões

Aplicações corporativas são soluções de software voltadas para a lógica de negócio de uma organização, empresa ou corporação. Elas frequentemente são desenvolvidas com gerenciamento centralizado e interagem com outros softwares corporativos. Tornou-se comum o desenvolvimento de soluções que envolvam o uso de objetos distribuídos e tecnologia do lado do servidor. Este trabalho monográfico apresentou as características de sistemas deste tipo desenvolvidos com tecnologia Java Enterprise Edition (Java EE), dando destaque aos serviços de segurança providos pela plataforma.

Este trabalho teve como foco o modelo de segurança desenvolvido para o Sistema Integrado de Apoio ao Paciente (SIAP) e como os serviços de segurança providos pela plataforma Java EE foram utilizados para implementar os mecanismos de segurança requeridos por este sistema. Ao ler este trabalho, um desenvolvedor Java EE encontrará orientações sobre como aplicar pelo menos uma das alternativas de uso dos serviços de segurança em um sistema Java EE.

O SIAP é um sistema voltado para a área da saúde e lida com informações sensíveis que precisam ser protegidas contra operações indevidas, muitos outros tipos de sistema também possuem necessidades como esta. Neste trabalho, discutiu-se desde as decisões que nortearam a política de segurança adotada até as medidas necessárias para impô-la. Descrições sobre como adicionar autenticação e autorização de forma declarativa por meio da inclusão de metadados em aplicações web e enterprise, além da auditoria programática, ativação da comunicação segura via protocolo HTTPS e a criação de um domínio customizado no servidor de aplicação, podem orientar desenvolvedores Java EE sobre o modo de utilizar as facilidades da plataforma face as inúmeras opções disponibilizadas por ela. Os serviços de segurança da plataforma Java EE são apenas parte da solução, a criptografia de assinatura digital e gerenciamento de chave pública encerram a garantia de segurança necessária as funcionalidades do SIAP.

Acreditamos que o estudo do modelo e mecanismos de segurança desenvolvidos para o projeto SIAP e descritos neste trabalho é útil na compreensão de todo o processo necessário a adição das primitivas de segurança em um sistema computacional, que aqui

foi descrito desde a definição de um modelo de segurança, com o estabelecimento de uma política, até detalhes da implementação dos mecanismos que auxiliaram na manutenção da segurança das informações mantidas pelo software do projeto. Através deste projeto, um desenvolvedor de aplicações corporativas conhecerá o percurso mínimo necessário para criar um sistema seguro, tendo como primeiro passo a definição de um modelo e como último a implementação dos mecanismos.

As principais contribuições deste trabalho foram:

- A descrição dos principais conceitos relacionados à segurança da informação;
- A especificação de um modelo de segurança adotado no projeto SIAP;
- A implementação dos mecanismos de segurança que auxiliaram na manutenção da segurança das informações mantidas pelo software do projeto SIAP;
- Apresentação das tecnologias utilizadas para a implementação dos mecanismos de segurança utilizados no projeto SIAP e uma descrição detalhada de como estes mecanismos foram implementados, servindo como guia ou roteiro para o desenvolvimento de aplicações distribuídas que requeiram o grau de segurança exigido em aplicações da área da saúde. Em particular, foram abordados:
  - A utilização dos mecanismos de autenticação e autorização da plataforma Java EE através da técnica declarativa com inclusão de metadados;
  - A manutenção de comunicação segura via protocolo HTTPS através do uso das facilidades oferecidas pela plataforma;
  - A implementação de um domínio customizado adequado as especificidades do SIAP. Este domínio demonstrou a versatilidade dos serviços de segurança da plataforma e do framework JAAS;
  - A implementação do processo de assinatura digital através da API XML Digital Signature;
  - O gerenciamento e validação de certificados digitais através do uso de APIs da JCA, como Java Certification Path, e de ferramentas como OpenSSL.

Referências Bibliográficas

[1] A. A. Tanenbaum e S. M. Van. . *Sistemas Distribuídos - Princípios e Paradigmas.* Editora Prentice Hall, 2 edição, 2008.

[2] R. Monson-Haefel e B. Burke. *Enterprise JavaBeans 3.0.* Editora Prentice Hall, 2 edição, 2008.

[3] E. Jendrock, R. Cervera-Navarro, I. Evans, K. Haase e W. Markito. *The Java EE 7 Tutorial.* Tutorial para o desenvolvimento de aplicações corporativas para a plataforma Java Enterprise Edition 7. Disponível em: http://docs.oracle.com/javaee/7/tutorial/doc/, acessado em 01 de julho de 2014, Abril de 2014.

[4] J. Kurose e K. Ross. *Redes de computadores e a Internet. Uma abordagem top-down.* Editora Pearson, 3 edição, 2006.

[5] W. Stallings. *Criptografia e Segurança de Redes.* Editora Pearson, 4 edição, 2006.

[6] A. A. Tanenbaum. *Redes de Computadores.* Editora Campus, 4 edição, 2003.

[7] F. J. da Silva e Silva. *Sistemas Distribuídos: Conceitos e Projeto. Introdução a Criptografia e Criptografia Simétrica.* Notas de aula. Disponível em: http://www.deinf.ufma.br/∼fssilva, acessado em 24 de outubro de 2013, Junho e 2013.

[8] F. J. da Silva e Silva. *Sistemas Distribuídos: Conceitos e Projeto. Criptografia Assimétrica e Funções Hash.* Notas de aula. Disponível em: http://www.deinf.ufma.br/∼fssilva, acessado em 24 de outubro 2013, Julho de 2013.

[9] Internet Engineering Task Force (IETF). *Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management.* Memorando sobre o gerenciamento de chaves através de certificado de chave pública. Disponível em: https://www.ietf.org/rfc/rfc1422, acessado em 13 de janeiro de 2014, Fevereiro de 1993.

[10] B. Leathem, L. Fryc e S. Rogers. *Develop applications using RichFaces 4.* Livro-Guia para o desenvolvimento com RichFaces 4. Disponível em: http://richfaces.jboss.org, acessado em 10 de julho de 2014.

[11] *Cryptography and SSL/TLS toolkit.* Documentação sobre o projeto OpenSSL. Disponível em: http://www.openssl.org, acessado em 10 de julho de 2014.

[12] M. L. da Silva. *Manual de Certificação para Sistemas de Registro Eletrônico em Saúde (S-RES).* SBIS/CFM, versão 4, 2013. Disponível em: http://www.sbis.org.br, acessado em 16 de julho de 2014, Dezembro de 2011.

[13] M. Coté. *JAAS in Action.* Coletânea sobre o framework JAAS. Disponível em: http://www.jaasbook.com, acessado em 23 de julho de 2014, Setembro de 2009.

[14] Ministério da Saúde. *Indicadores e Dados Básicos - Brasil - 2012 (IDB-2012).* Tabela sobre proporção de óbitos (em %) por grupos de causas segundo a unidade da federação em 2011 (dados da Rede Interagencial de Informações para a Saúde - RIPSA). Disponível em: http://tabnet.datasus.gov.br/cgi/idb2012/matriz.htm, acessado em 09 de agosto de 2014.

[15] Oracle. *XML Digital Signature API.* Documentação Java SE sobre a API XML Digital Signature. Disponível em:

http://docs.oracle.com/javase/6/docs/technotes/guides/security/xmldsig/

XMLDigitalSignature.html, acessado em 09 de agosto de 2014.

[16] Oracle. *JAAS Tutorials.* Documentação Java sobre o framework JAAS. Disponível em: http://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/tutorials/, acessado em 09 de agosto de 2014.

[17] Oracle. *Oracle GlassFish Server 3.1 Application Development Guide.* Manual do desenvolvedor de aplicações Java EE com o servidor GlassFish. Disponível em: http://docs.oracle.com/cd/E18930_01/html/821-2418/, acessado em 09 de agosto de 2014, Julho de 2011.