UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Marlon Trapp

UMA ALTERNATIVA AOS APPLETS EM JAVA PARA ACESSO A DISPOSITIVOS

Florianópolis

2016

Marlon Trapp

UMA ALTERNATIVA AOS APPLETS EM JAVA PARA ACESSO A DISPOSITIVOS

Trabalho de conclusão de curso submetida ao Programa de Graduação em Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. Jean Everson Martina

Florianópolis

2016

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

Marlon Trapp

UMA ALTERNATIVA AOS APPLETS EM JAVA PARA ACESSO A DISPOSITIVOS

Esta Trabalho de conclusão de curso foi julgada aprovada para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovada em sua forma final pelo Programa de Graduação em Ciências da Computação.

Florianópolis, 01 de julho 2016.

Prof. Dr. Mario Antonio Ribeiro Dantas
Coordenador do Curso

Banca Examinadora:

Me. Lucas Pandolfo Perin

Prof. Dr. Jean Everson Martina
Orientador

Me. Rick Lopes

RESUMO

Com a descontinuação do plugin Java Applet alguns sistemas web que se utilizavam dessa tecnologia para acessar dispositivos ficaram sem alternativas viáveis para a sua substituição. Neste trabalho busca-se estudar e propor uma alternativa aos Java Applets para acesso a dispositivos em aplicações web. Ao decorrer do trabalho foi definido um modelo que tem a capacidade de substituir os applets e oferecer acesso aos dispositivos conectados no computador do usuário. O modelo proposto utiliza-se de uma modelagem modular a qual permite conectar outros módulos que possam prover acesso a qualquer tipo de dispositivo. Posteriormente esse modelo foi aplicado, o qual resultou no desenvolvido de um protótipo chamado de *Device Server* e um módulo com o intuito de testar a solução.

Palavras-chave: Java Applet. acesso a dispositivos. aplicações web.

ABSTRACT

With the descontinuation of the Java Applet plugin some web systems that used that plugin to access devices are without viable alternatives. In this work we study and look for an alternative to Java Applets for access devices in web applications. In the course of this work was defined a model that has the capacity of substitute the applets and offer access to the devices connected in the user's computer. The proposed model uses a modular design that allow connect another modules that can provide access to any kind of device. Finally this model was applied, this results in the development of a prototype called *Device Server* and a module to test the solution.

Keywords: Java Applet. devices access. web applications.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Com a popularização da internet a partir da metade dos anos 90 algumas empresas começaram a migrar para o novo ambiente virtual. No começo, se beneficiavam da possibilidade de digitalizar seus processos e informações e armazená-los remotamente, sendo acessíveis de qualquer lugar através da internet.

Com o passar do tempo e a expansão da internet as instituições e pessoas começaram a migrar tarefas cotidianas para o ambiente eletrônico. Um exemplo prático foi a criação do *internet banking*, o qual facilitou e agilizou processos que antes precisavam ser fisicamente executados dentro de uma agência. Mais recentemente ainda temos as lojas virtuais as quais tiveram um crescimento expressivo durante a última década.

Uma aplicação web é uma aplicação que é executada na internet através de um navegador. Ou seja, o processamento e armazenamento dos dados da aplicação ficam na web. Estas aplicações normalmente não precisam ser instaladas no computador do usuário. Alguns exemplos de aplicações web são clientes de e-mail e editores de arquivos online.

A aplicação web algumas vezes requer acesso a recursos da maquina onde está sendo executada, por exemplo um chat online que precisa acesso ao microfone e a câmera do computador. Nesse caso é facilmente atendida pois a tecnologia HTML5 e os navegadores suportam esse tipo de dispositivos. Porém quando a aplicação necessita acesso a um dispositivo menos usual, ou até proprietário, esse quadro se complica pois os navegadores não provêm esse suporte nativamente.

O acesso a dispositivos conectados ao computador do usuário através de navegadores usualmente pode ser feito de três maneiras. Primeira, por suporte do próprio navegador ao dispositivo, como acontece atualmente com webcams. Segunda, por extensões específicas para aquele navegador. E terceira, por meio de plugins de terceiros.

Dar suporte a uma classe de dispositivos utilizando a primeira solução é eficaz, porém precisa-se convencer a instituição desenvolvedora do navegador a fazê-lo. Além disso, essas instituições não irão se interessar em suportar um dispositivo que não é usado pela grande parte de seus usuários. Partimos então para a segunda opção, desenvolver uma extensão para o navegador. Essa alternativa também é interessante, porém necessitaria o desenvolvimento de uma extensão

para cada combinação de SO e navegador, o que torna essa alternativa custosa para se desenvolver e manter. A terceira é a alternativa que se torna menos custosa, pois passa-se a responsabilidade da camada que executa no navegador para o plugin utilizado, e pode focar-se apenas em suportar diferentes SO's.

Um dos navegadores mais utilizados atualmente, o Google Chrome (NETMARKETSHARE, 2016), está removendo o suporte a plugins implementados com a tecnologia *Netscape Plugin Application Programming Interface* (NPAPI) por questões de desempenho e segurança (SCHUH, 2014). O NPAPI é uma *application programming interface* (API) que permite a criação de plugins para navegadores web. Ele foi criado em 1995 pela *Netscape Communications* e implementado primeiramente no navegador de mesmo nome, o Netscape, depois adotado por vários outros navegadores.

Um plugin utilizado para se fazer o acesso a dispositivos em navegadores é o Java Applet, ele carrega uma aplicação Java que interage com o navegador no momento em que o usuário acessa a página, tendo apenas as limitações da própria linguagem. Porém esse plugin utiliza a NPAPI como base e com a descontinuação do suporte à essa API ele não será mais suportado pelo navegador mais utilizado atualmente. Segundo informações publicadas no blog da Oracle o Java 9 não deve mais trazer o plugin junto à plataforma (TOPIC, 2016)

1.1 JUSTIFICATIVA

No Laboratório de Segurança em Computação (LabSEC) da Universidade Federal de Santa Catarina (UFSC) utiliza-se um Java applet no projeto Hawa, um projeto de Autoridade Certificadora online. Esse applet é responsável por fazer a comunicação com dispositivos criptográficos conectados ao computador do usuário.

Com o anúncio da Oracle sobre a descontinuação do plugin, é interessante a análise de alguma alternativa para a substituição desse Java applet. E ainda, futuramente, possivelmente será necessário o o acesso a outros dispositivos como leitores biométricos.

1.2 OBJETIVOS

Abaixo serão apresentados os objetivos deste trabalho.

1.2.1 Objetivo geral

O objetivo desse trabalho é fornecer uma alternativa ao plugin do Java Applet para acesso a dispositivos. Essa alternativa deve ter baixa dependência do navegador e da plataforma. Espera-se ainda que a solução seja segura, evitando ataques comuns para aplicações web como o *man-in-the-middle*, acesso não autorizado entre outras que possam prejudicar o usuário ou desenvolverdor.

1.2.2 Objetivos específicos

Como objetivos específicos espera-se:

1. Utilizar somente tecnologias já existentes em navegadores atuais para diminuir as dependências do projeto.
2. Minimizar as decisões necessárias por parte do usuário que muitas vezes não tem conhecimento técnico necessário para fazer configurações avançadas.
3. Deixar a solução livre de ataques do tipo man-in-the-middle, e prover sigilo em sua comunicação.

2 FUNDAMENTAÇÃO TEÓRICA

Aqui são apresentados os conceitos fundamentais para o entendimento do trabalho.

2.1 CONCEITOS BÁSICOS DE SEGURANÇA

De acordo com Menezes, Oorschot e Vanstone (2001), para entender criptografia um entendimento dos problemas relacionados à segurança da informação em geral é necessário. A segurança da informação se manifesta de várias formas de acordo com a situação e requisitos. Independente de quem está envolvido, em certo nível, todas as partes da transação devem ter confiança de que certos objetivos associados à segurança da informação foram alcançados. Alguns objetivos necessários para o entendimento do trabalho são os seguintes:

Confidencialidade: manter a informação secreta a todos, exceto para os autorizados a vê-la.

Integridade: assegurar de que a informação não foi alterada por alguém sem autorização ou por meios desconhecidos.

Autenticação ou identificação: comprovar a identidade de uma entidade (uma pessoa, um computador, um cartão de cŕedito, etc.).

Autenticação de mensagem: comprovar a fonte da informação.

Assinatura: um meio de vincular a informação a uma entidade.

Autorização: sanção oficial para outra entidade fazer ou ser algo.

Controle de Acesso: restringe o acesso a recursos às entidades privilegiadas.

Certificação: garantia da informação por uma entidade confiável.

Não-Repúdio: evita a recusa de autorizações prévias ou ações.

Revogação: retração de certificação ou autorização.

2.2 CRIPTOGRAFIA

Vamos supor que Alice quer enviar uma mensagem ao Bob, ela usa um canal qualquer para o envio, porém ela não controla esse canal de comunicação. Eve, alguém que está observando o canal consegue ler a mensagem que contem informações sensíveis, que somente Bob devia ter acesso (Fig. (1)). Como Alice pode enviar a mensagem ao Bob sem precisar se preocupar se alguém está observando esse canal? Para solucionar esse problema usa-se a criptografia, do Grego kryptós "escondido", mais graphé, "escrita", ou seja, escrita secreta ou escondida. De acordo com Housley e Polk (2001) criptografia é a técnica de cifrar, e decifrar, mensagens privadas. Uma mensagem é cifrada para manter seu conteúdo privado ou para proteger sua *confidencialidade*. A criptografia moderna provê técnicas que conseguem também determinar quando uma mensagem foi modificada e identificar o autor da mensagem. Uma mensagem inalterada provê *integridade* e o conhecimento de origem provê a *autenticação de mensagem*.

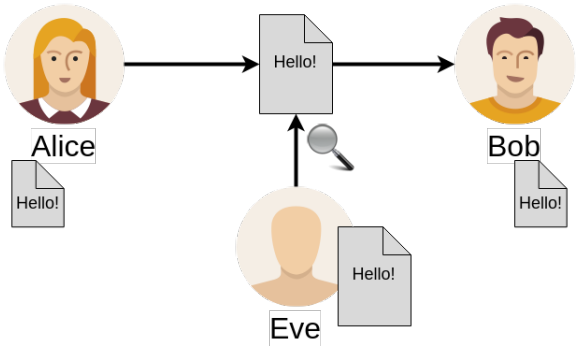

Figura 1 – Comunicação sem uso de criptografia.

2.2.1 Criptografia simétrica

Na criptografia simétrica, tanto o remetente quanto o receptor, ou Alice e Bob, possuem uma chave, chamada também de chave secreta.

Nesse modelo (Fig. (2)) quando Alice quer enviar uma mensagem ao Bob ela utiliza algum algoritmo criptográfico combinado com a chave que ambos possuam e envia a mensagem cifrada a Bob. Quando Bob receber, em posse da mesma chave e utilizando o mesmo algoritmo, executa a decifragem da mensagem e chega à mensagem original. Nesse modelo se Eve tentar observar a mensagem no canal de comunicação ela verá somente dados aleatórios que não fazem sentido sem a chave e o algoritmo usado para cifragem.

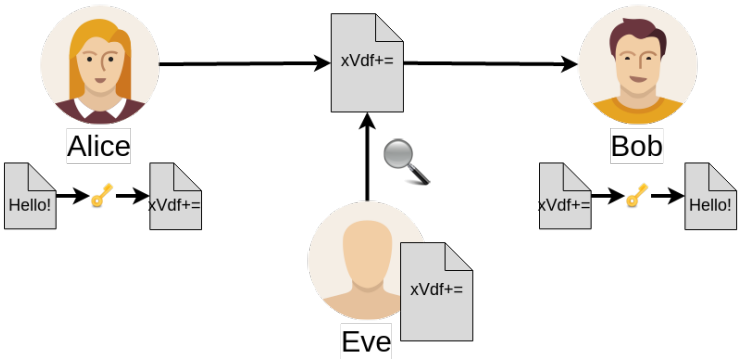

Figura 2 – Comunicação com criptografia simétrica.

2.2.2 Criptografia assimétrica

Na criptografia assimétrica cada parte possui um par de chaves, uma chave pública e uma privada. Tudo que é cifrado com a chave pública só é decifrado com sua chave privada correspondente e vice e versa. A chave pública como o próprio nome diz é pública, então pode ser distribuída livremente. Já a chave privada deve permanecer com o respectivo dono. Na Fig. (3) podemos ver como seria uma comunicação criptografada com criptografia assimétrica. Caso Alice queira enviar uma mensagem para Bob ela utilizaria a chave pública dele para cifrar o conteúdo da mensagem e enviar a mensagem cifrada. Assim quando Bob receber a mensagem ele pode decifrar a mensagem cifrada com sua chave privada, como só Bob tem acesso a sua chave privada Eve não terá como decifrar a mensagem e chegar à mensagem em claro.

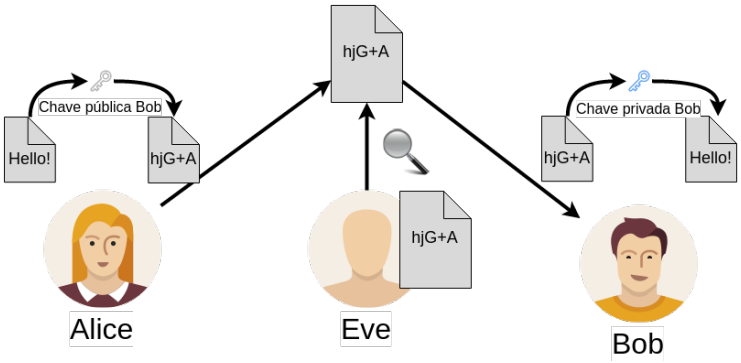

Figura 3 – Comunicação com criptografia assimétrica.

2.3 RESUMO CRIPTOGRÁFICO

O resumo criptográfico é um valor gerado por uma função matemática que recebe uma entrada de dados de tamanho variável e converte essa entrada em uma saída com tamanho fixo, geralmente menor que a entrada. O objetivo dessa função é gerar uma espécie de impressão digital do valor de entrada, também conhecido como *hash*, com isso podemos obter uma resposta estatística de quando um resumo criptográfico, ou *hash*, pertence a um determinado dado.

Uma função de resumo criptográfico deve ser unidirecional: é fácil computar o *hash* a partir da entrada, porém é extremamente custoso criar o dado de entrada na posse somente de um determinado *hash*. Uma boa função de resumo criptográfico também é livre de colisão, ou seja, duas entradas diferentes dificilmente gerarão um mesmo resumo criptográfico.

A função de resumo criptográfico é pública, não há segredos no processo. A segurança nessa operação é alcançada pela sua não reversibilidade. A saída não depende da entrada de nenhuma maneira dedutível. Um único *bit* diferente no dado de entrada gera um resumo com aproximadamente metade dos *bits* diferente que o original. Dado um valor *hash* é computacionalmente inviável gerar os dados que correspondem a esse resumo criptográfico (SCHNEIER, 1996, 2, p. 30).

2.4 DIFFIE HELLMAN

Diffie-Hellman foi o primeiro algoritmo de chaves públicas inventado, em 1976. Ele se utiliza da dificuldade em calcular logaritmos discretos em um corpo finito, comparado a facilidade de se computar operações de exponenciação no mesmo corpo. *Diffie-Hellman* também pode ser utilizado para a distribuição de chaves - Alice e Bob podem usar esse algoritmo para gerar uma chave secreta - porém não pode ser utilizado para cifragem ou decifragem (SCHNEIER, 1996, 22, p. 513).

O algoritmo de troca de chaves de *Diffie-Hellman* é executado da seguinte maneira, primeiramente se têm dois números públicos conhecidos: um número primo p e um inteiro g que é uma raiz primitiva de p. Supondo que os usuários Alice e Bob queiram fazer o acordo de troca de chave. Alice escolhe um número inteiro randômico X_a tal que $X_a < p$ e calcula $Y_a = g^{X_a} \bmod p$. De modo similar, mas independente, Bob escolhe um número inteiro randômico X_b tal que $X_b < p$ e calcula $Y_b = g^{X_b} \bmod p$. Cada lado mantem seu X privado e envia seu Y publicamente para o outro lado. Então o usuário A calcula a chave $K = (Y_b)^{X_a} \bmod p$ e o usuário B calcula a chave $K' = (Y_a)^{X_b} \bmod p$. K e K' são idênticas e agora podem ser usadas como a chave secreta para as duas partes se comunicarem, abaixo um exemplo do processo:

1. Vamos tomar $p = 5$ e $g = 3$;

2. Alice escolhe $X_a = 2$ e calcula $Y_a = 3^2 \bmod 5$, então $Y_a = 4$;
   Bob escolhe $X_b = 4$ e calcula $Y_b = 3^4 \bmod 5$, então $Y_b = 1$;

3. Alice envia $Y_a = 4$ para Bob;
   Bob envia $Y_b = 1$ para Alice;

4. Alice em posse de $Y_b = 1$ calcula $K = (1)^2 \bmod 5$, $K = 1$;
   Bob em posse de $Y_a = 4$ calcula $K' = (4)^4 \bmod 5$, $K' = 1$;

5. Alice e Bob agora possuem uma chave secreta acordada, $K = K' = 1$;

Como X_a e X_b são privados, um atacante tem somente os seguintes dados para trabalhar: p, g, Y_a e Y_b. O atacante então é forçado a calcular o logaritmo discreto para determinar a chave secreta trocada por A e B. Como mencionado anteriormente é muito difícil se calcular o logaritmo discreto, para primos muito grandes essa tarefa é impraticável computacionalmente (STALLINGS, 2003, 10, 295).

2.5 SECURE SOCKET LAYER AND TRANSPORT LAYER SECURITY

O *Secure Socket Layer* (SSL) é um protocolo que foi planejado para prover um serviço fim-a-fim seguro e confiável utilizando o protocolo TCP.

O protocolo SSL provê serviços básico de segurança para vários protocolos de mais alto nível. Em particular o HTTP, que propicia o serviço de transferência entre o cliente Web e servidor. Três protocolos de alto nível são definidos como parte do SSL: o protocolo de *handshake*, de troca de cifras e o protocolo de alerta.

Quando esse protocolo foi submetido a Internet Engineering Task Force (IETF) passou a ser chamado de *Transport Layer Security* (TLS), o grupo de trabalho TLS desenvolveu a primeira versão do TLS. A primeira versão publicada pode ser vista essencialmente como o SSLv3.1 e é muito próximo e mantém retrocompatibilidade com o SSLv3. (STALLINGS, 2003, 16, 488).

2.6 ASSINATURA DIGITAL

Um uso importante da criptografia assimétrica, ou criptografia de chave pública, é que ela provê mecanismos necessários para a base da assinatura digital. A chave privada é utilizada para gerar a assinatura e a chave pública para validá-la. Em aplicações reais as mensagens não são assinadas diretamente, mas sim utilizada uma função de resumo criptográfico para gerar seu *hash* e aí sim esse *hash* é assinado pela chave privada (Fig. (4)).

A assinatura digital pode prover evidências importantes em uma disputa. Caso Alice use sua chave privada para assinar uma mensagem, Bob pode validar a assinatura com a chave pública de Alice (Fig. (5)). Como Bob não precisa da chave privada de Alice para validar a assinatura, ele não tem a informação necessária para gerar uma assinatura válida com a mensagem alterada (HOUSLEY; POLK, 2001, 2,12).

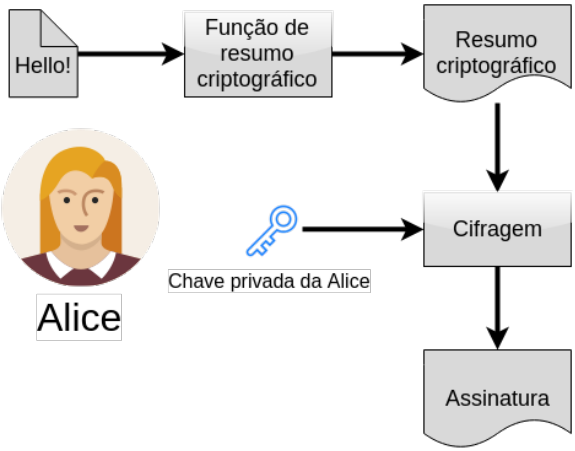

Figura 4 – Processo de assinatura digital.

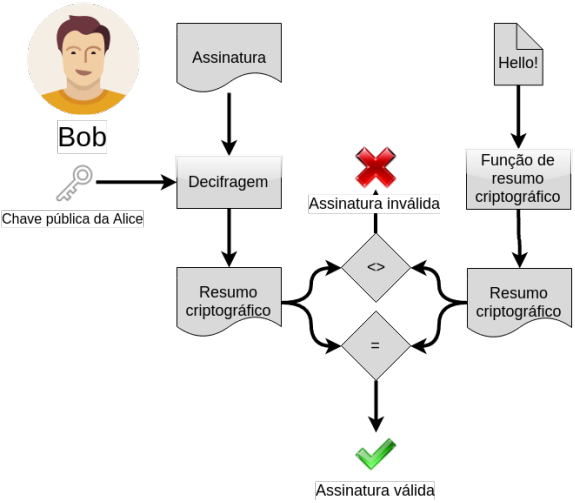

Figura 5 – Processo de verificação de uma assinatura digital.

2.7 CERTIFICADO DIGITAL

A criptografia assimétrica, ou criptografia de chave pública, envolve o uso do par de chaves pública/privada para facilitar o utilização de assinaturas digitais e serviços de gerenciamento de chaves. O princípio fundamental que possibilita essa tecnologia de ser usada em larga escala é a possibilidade da livre distribuição da chave pública entre as entidades que precisam desse componente. Porém, a distribuição da chave pública sem alguma forma de garantir a integridade iria comprometer a base desses serviços, então percebeu-se que um método escalável e seguro seria necessário para a transmissão da chave pública para as partes que necessitam dela.

O certificado digital foi uma das alternativas para essa distribuição, ele liga o nome de uma entidade (e possivelmente atributos adicionais associados com essa entidade) com a chave pública correspondente (ADAMS; LLOYD, 2003, 6, 70). O certificado ideal segundo Housley e Polk (2001) deveria possuir as seguintes nove propriedades:

1. Ser um objeto puramente digital, podendo então ser distribuído pela internet e processado automaticamente;

2. Conter o nome do usuário que detém a chave privada, identificação da organização ou companhia do usuário, e inclui informações de contato;

3. Ser fácil de determinar se o certificado foi emitido recentemente;

4. Ser criado por uma autoridade confiável ao invés do detentor da chave privada;

5. Dado que a autoridade confiável pode criar vários certificados, até mesmo para um mesmo usuário, deveria ser fácil diferenciá-los;

6. Ser fácil de determinar se um certificado é autentico ou foi forjado;

7. Ser a prova de modificações, para que ninguém possa alterar o seu conteúdo;

8. Poder identificar imediatamente quando a informação em nosso certificado não correspondem mais com a atualidade;

9. Poder determinar a partir do certificado quais os usos permitidos para ele.

Uma aproximação desse certificado ideal existe hoje em dia. É chamado de certificado de chave pública, referindo-se aqui ao certificado X.509 versão 3, o qual é definido pela *Request for Comments* (RFC) 3280. Um certificado de chave pública é um objeto puramente digital. O certificado de chave pública contém campos com o nome do usuário juntamente com sua chave pública. Esse certificado pode também indicar a organização ou companhia do usuário, e pode incluir informações de contato também. Além disso ele inclui também dois campos de data, que indicam a data de ativação e de expiração do certificado. O certificado também possui o nome da autoridade confiável que o criou, também chamada de emissor. O emissor adiciona um número serial em cada certificado criado para que possam ser facilmente identificados.

Ao final da montagem da estrutura do certificado o conteúdo inteiro é protegido através da assinatura digital do emissor. O emissor trata o conteúdo do certificado como uma mensagem e gera uma assinatura digital a qual é anexada ao certificado final.

2.8 LISTA DE CERTIFICADOS REVOGADOS

Certificados uma vez que são distribuídos tornam-se praticamente impossíveis de serem recolhidos todos novamente. De fato, o problema é pior para certificados digitais. Já que são objetos puramente digitais, podem ser facilmente copiados e redistribuídos.

Dessa forma o emissor do certificado deve prover a informação necessária para informar se um determinado certificado foi revogado, por qualquer motivo, por exemplo o comprometimento da chave privada do usuário. A forma encontrada para fazer essa tarefa com certificados foi a de criação de uma LCR (lista de certificados revogados). A LCR contém uma lista de números seriais de certificados não expirados que foram revogados e então não devem ser considerados confiáveis. A LCR é um objeto digital, que deve ser distribuído para quem quiser verificar o estado de revogação de um certificado. A LCR garante sua integridade através da assinatura com a chave privada do emissor, assim como os certificados. O emissor adiciona a data de emissão e expiração da LCR para que quem consultá-la tenha certeza que está consultando dados atualizados.

2.9 INFRAESTRUTURA DE CHAVES PÚBLICAS

Uma infraestrutura de chaves públicas (ICP) é planejada para facilitar o uso da criptografia assimétrica. Uma ICP alcança seus objetivos através da criação de certificados de chave pública e listas de certificados revogados (LCRs).

É difícil construir um único componente que pode seguramente criar e distribuir certificados e LCRs. ICPs são construídas a partir de vários componentes, cada um planejado para realizar tarefas particularmente bem.

2.9.1 Autoridade Certificadora

A autoridade certificadora (AC) é a peça principal de uma ICP. De acordo com Housley e Polk (2001) uma AC é constituída do seu hardware computacional, software e as pessoas que a operam. A AC é conhecida através de dois atributos: seu nome e sua chave pública. A AC realiza quatro operações básicas em uma ICP:

- Emitir certificados (criar e assiná-los).
- Manter o estado dos certificados e emitir LCRs.
- Publicar os certificados não expirados e LCRs para que os usuários possam obter a informação que eles precisam para implementar serviços de segurança.
- Manter arquivado as informações sobre a expiração ou revogação de certificados que foram emitidos por ela.

Esses requerimentos podem ser difíceis de satisfazer simultaneamente, por isso para cumprir esses requerimentos a AC pode delegar algumas tarefas a outros componentes da infraestrutura.

2.9.2 Autoridade Registradora

Uma AR (Autoridade Registradora) é planejada afim de verificar o conteúdo dos certificados para a AC. O conteúdo do certificado pode refletir informações apresentadas pela entidade que requisitou o certificado, como a carteira de motorista, CPF, etc.

Como a AC, a AR é a coleção de hardware, software, e pessoas que a operam. Cada AC mantém uma ou mais ARs, ou seja, uma lista de ARs que sejam consideradas confiáveis pela AC. Cada AR é conhecida pela AC pelo seu nome e chave pública. Verificando a assinatura da AR em uma mensagem a AC pode assegurar que a AR credenciada foi quem proveu a informação.

2.9.3 Arquiteturas de infraestruturas de chaves públicas

Existem várias tipos de arquiteturas que ICPs podem seguir, nesse trabalho abordaremos somente as mais simples e necessárias para o entendimento do mesmo.

2.9.3.1 Autoridade certificadora única

Nesse modelo, exemplificado na Fig. (6), existe uma AC que emite certificados e LCRs somente para usuários finais. Ela não emite certificado para outra autoridade certificadora. Assim um certificado dentro dessa ICP é fácilmente validado, pois o caminho de certificação consiste somente em um certificado de AC e o certificado do usuário final.

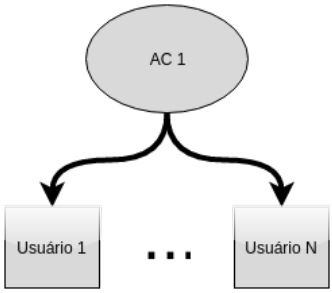

Figura 6 – Arquitetura de AC única.

2.9.3.2 Autoridades certificadoras hierárquicas

Aqui temos uma AC raíz da qual toda a cadeia de ACs deriva (Fig. (7)). A AC raíz emite certificados para outras ACs, que podem ser finais, as quais emitem certificados para usuários finais, ou intermediárias que emitem certificados para outras ACs.

A validação de um certificado emitido em uma estrutura hierárquica consiste da validação de todos os certificados acima do certificado do usuário até alcançar-se a AC raíz.

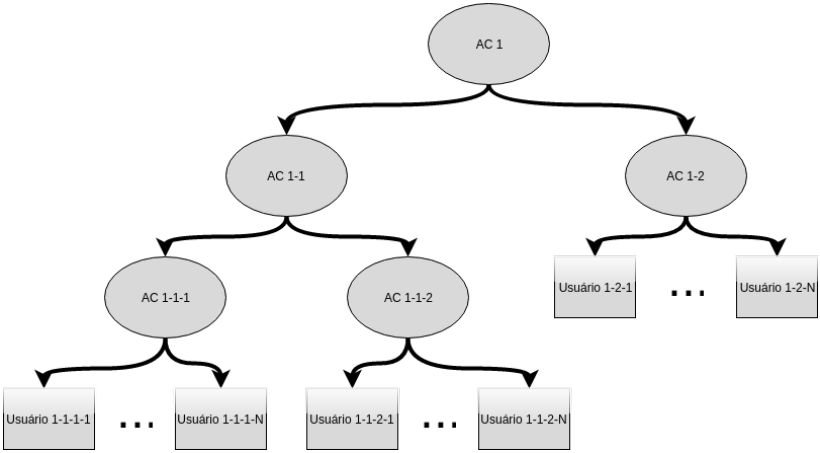

Figura 7 – Arquitetura hierárquica.

3 PROPOSTA

No início do trabalho procurou-se uma forma de qualquer navegador se comunicar com algum artefato que tenha acesso aos dispositivos conectados ao computador do usuário, resolvemos analisar o que todos os navegadores têm em comum. Duas tecnologias básicas que estão em todos os navegadores são o suporte ao protocolo HTTP (*Hypertext Transfer Protocol*) e a scripts em JavaScript. Trabalhando a partir dessas duas tecnologias começou-se a criar o modelo de proposta.

3.1 CONEXÕES

Com a tecnologia HTTP e JavaScript podemos executar quase tudo o que o navegador pode fazer, dentre elas acessar recursos em servidores, se esse servidor estiver na máquina do cliente então temos nosso artefato com acesso aos dispositivos conectados. Doravante chamaremos esse servidor de *Device Server*, ele estará instalado no computador do cliente, a arquitetura por parte do cliente pode ser vista na Fig. (8). A partir da comunicação do navegador com o dispositivo começou-se a

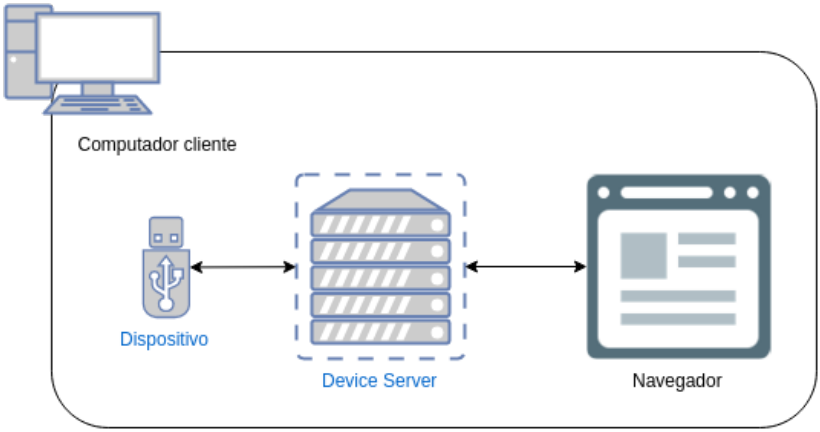

Figura 8 – Modelo da solução por parte do cliente.

analisar as alternativas para a comunicação com o aplicativo web que

utilizará o dispositivo. O objetivo era que a solução seja igualmente, fácil de utilizar em um aplicativo web, e seja segura, não expondo os dados de dispositivo do usuário. Chegou-se a 2 modelos básicos de comunicação. O primeiro, mostrado na Fig. (9), consiste em uma comunicação direta do *Device Server* com o servidor web. Nesse modelo após o carregamento da página através da conexão 1, o navegador requisita a criação de uma sessão com o *Device Server*, pela conexão 2, informando o endereço do servidor web, assim os dois pontos se comunicam diretamente através da conexão 3. O segundo modelo, Fig. (10),

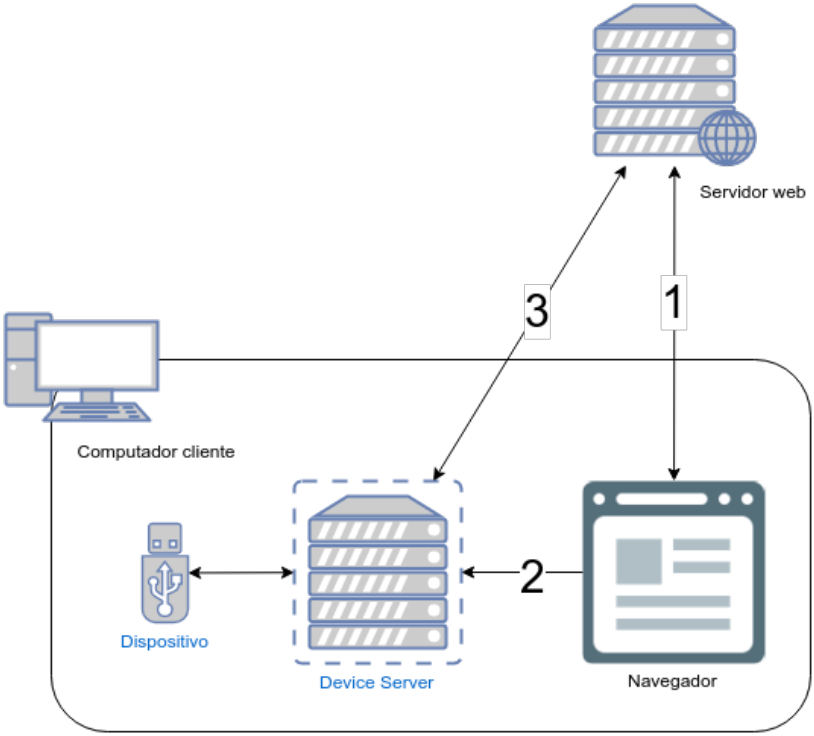

Figura 9 – Modelo de conexão direta.

não se comunica diretamente com o servidor web, toda a comunicação passa pelo navegador. Então, depois do carregamento da página pela conexão 1, o navegador requisita a criação de uma sessão com o *Device Server* através da conexão 2, o *Device Server* irá responder novamente pela conexão 2 ao navegador que encaminhará tudo para o servidor web

através da conexão 1.

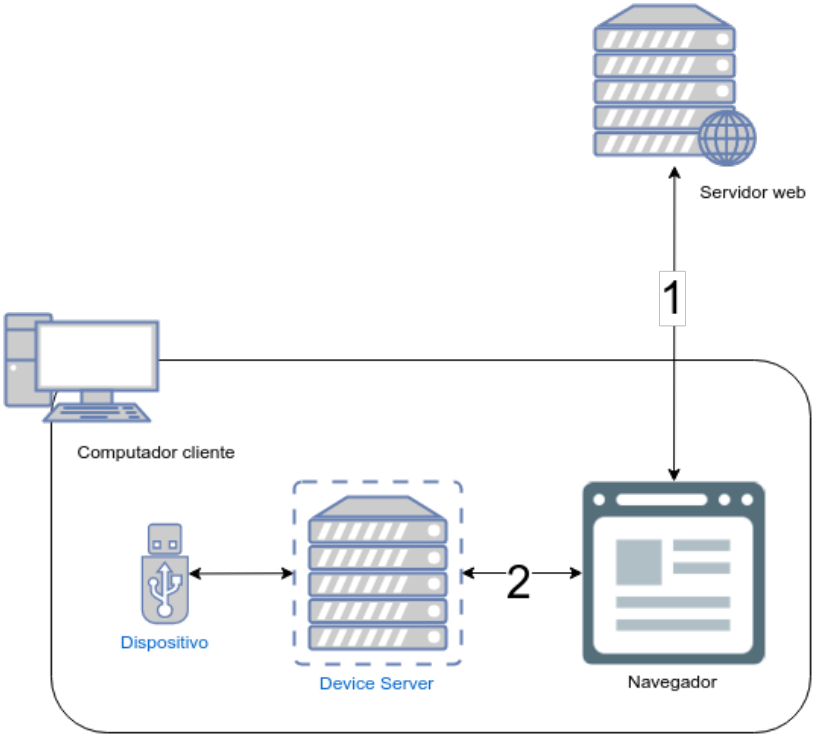

Figura 10 – Modelo de conexão indireta.

O primeiro modelo diminuiria a latência entre a comunicação do *Device Server* e o servidor web, porém abriría-se um vetor de ataque no computador do usuário, pois como o *Device Server* precisaria ser acessado diretamente da rede externa agentes maliciosos poderiam se utilizar disso para explorar alguma possível brecha. A comunicação entre o *Device Server* e o servidor web teria que ser padronizada, e a aplicação que quisesse utiliza-lo teria que respeitar esse padrão. Já o segundo modelo precisará de uma intervenção do navegador para cada comunicação feita com o *Device Server*. Apesar do custo, isso deixará a solução menos passível de ataques, pois o *Device Server* receberá requisições somente no localhost. Na segunda solução, como o navegador intermedeia a conexão, fica a cargo dos desenvolvedores da aplicação escolher como serão repassado as informações ao servidor web, por

AJAX (*Asynchronous Javascript and XML*), REST (*Representational State Transfer*), SOAP (*Simple Object Access Protocol*), etc. Como o foco primário da solução é segurança decidimos por utilizar o segundo modelo, com conexão indireta.

3.2 PROTOCOLOS E PADRÕES DE COMUNICAÇÃO

Definido o modelo de conexões, passou-se à escolha de como cada ponta se comunicará com a outra. Com base na Fig. (10), a conexão 1 fica a critério do desenvolvedor da aplicação web, já a conexão 2, entre o navegador e o *Device Server* precisa ser especificada.

Levantou-se duas alternativas, *websockets* ou REST. Para a solução de acesso a dispositivos a alternativa mais interessante seria a utilização de *websockets*, pois ela permite comunicação assíncrona e ainda mais rápida do que com REST, porém como é uma especificação recente e somente os navegadores mais atualizados a suportam optou-se pela utilização do REST. O padrão REST já está bem estabelecido e já existem bibliotecas que facilitam muito sua implementação.

3.3 SEGURAÇA DA SOLUÇÃO

Como o projeto interage com dispositivos do usuário os quais podem conter informações sensíveis, como dispositivos criptográficos, focou-se na questão segurança. O projeto se preocupa tanto em autenticar quem está requisitando acesso ao dispositivo quanto a como os dados devem trafegar até o servidor web.

3.3.1 Autenticação

A autenticação é feita através de certificados digitais, ou seja, teremos uma AC (autoridade certificadora) que emitirá certificados às entidades ou aplicações que desejam acesso ao *Device Server*. Com isso toda a aplicação terá acesso a um par de chaves atrelado a um certificado o qual será usado para fazer sua autenticação. Conforme a Fig. (11) podemos ver como funcionará na prática o esquema de autenticação. No passo 1, quando o navegador requisita a página, o servidor web retorna-a e junto envia o certificado para acesso ao *Device Server*. Assim a página carregada no navegador poderá requisitar a abertura de uma sessão com o *Device Server*, mostrada no passo 2,

passando o certificado da aplicação e o módulo a qual deseja acesso. O *Device Server* verifica se o certificado é valido e se não está revogado através de uma LCR (Lista de certificados revogados). Passando nessas duas verificações o *Device Server* começa a execução do protocolo TLS, assim gerando um canal de comunicação seguro entre ele e o servidor web, passo 3 na Fig. (11). Estabelecido o protocolo o *Device Server* liberará o acesso ao módulo requisitado através dessa sessão.

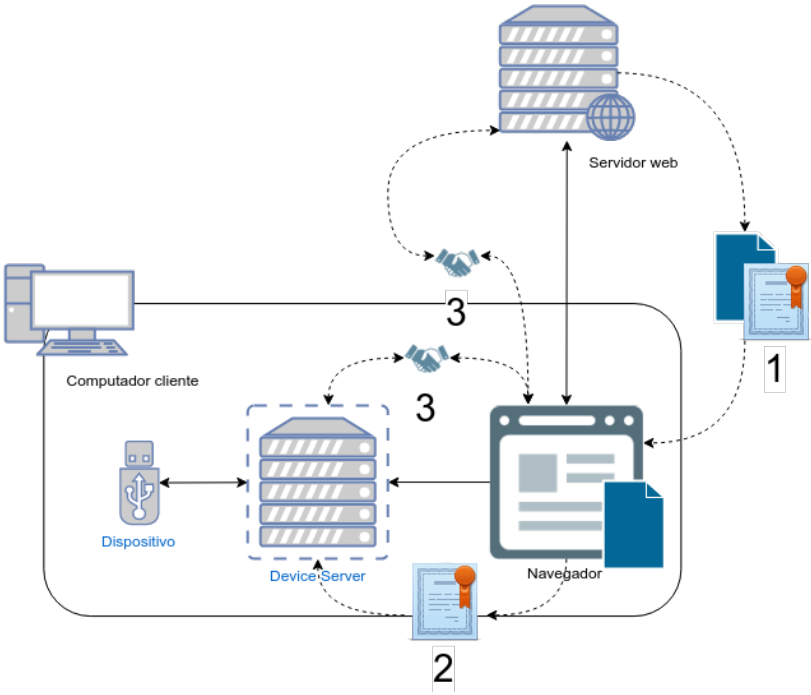

Figura 11 – Esquema de autenticação.

3.3.2 Sigilo

O sigilo é alcançado cifrando as mensagens passadas de uma ponta a outra, podería-se utilizar as chaves atreladas ao certificado usado no passo de autenticação para cifrar e decifrar mensagens, porém como sabemos (SCHNEIER, 1996, 10, p. 216) o uso de criptografia

assimétrica para cifragem de decifragem é menos eficiente que o uso de criptografia simétrica. Por isso escolhemos, ao invés de usar as chaves dos certificados gerados para a autenticação, executar o protocolo TLS (*Transport Layer Security*) usado na internet para garantir que o site que está sendo acessado é realmente quem diz ser. Então, usando novamente a Fig. (4), ao final do passo 3, o servidor web e o *Device Server* terão uma conexão por um tunel TLS. Assim podemos usar essa conexão sem nos preocuparmos se o meio está comprometido, por exemplo, se houver uma brecha no navegador que possibilite que uma página acesse o conteúdo de outra aberta ao mesmo tempo. Com o uso do TLS impossibilita-se o ataque, pois mesmo que a página maliciosa consiga saber qual a sessão, ela não conseguira transmitir mensagens que façam sentido quando o *Device Server* decifrá-las com a chave de gerada pelo protocolo TLS. Dessa maneira estará evitando-se os ataque do tipo *man-in-the-middle*.

4 PROTÓTIPO

O protótipo foi desenvolvido com a linguagem de programação Java, para alcançar múltiplas plataformas sem a necessidade de reescrita de código. Além do protótipo do *Device Server* foram desenvolvidos um módulo de teste para ser gerenciado pelo *Device Server* e uma pequena aplicação web, escrita totalmente em JavaScript e HTML, como prova de conceito.

4.1 DEVICE SERVER

O desenvolvimento começou pela ideia base da proposta, a criação de um servidor local para a comunicação com o navegador. Para isso se tornar mais fácil procurou-se por implementações de servidores web em Java que suportassem a adição do Java API for RESTful Web Services (JAX-RS), especificada pela JSR 339, a qual ajuda o desenvolvimento de serviços REST em java usando anotações para simplificar o processo. Dentre as várias opções de projetos analisados chegou-se ao Jetty e ao Grizzle. Ambos utilizam o Jersey para prover as funcionalidades do JAX-RS. Após o teste das soluções, notou-se que ambas solucionavam o problema de criar um servidor local, porém por questões de segurança decidiu-se utilizar o Grizzle pela possibilidade da configuração para ouvir somente requisições vindas de localhost, enquanto o Jetty não oferecia essa opção, tendo que implementar filtros para recusar conexões externas.

Uma simplificação do modelo que foi aplicada no protótipo foi a troca do protocolo TLS pela execução do protocolo de *Diffie-Hellman*. O qual tornou a implementação mais simples, mas ainda assegurando certa segurança a implementação.

Para criar o *Device Server* analisou-se os recursos necessários para o seu funcionamento, os recursos que foram definidos como necessários estão listados na Tab. (1).

O primeiro método, método GET na raiz do recurso do *Device Server*, lista todos os módulos carregados no *Device Server*. Para estabelecimento da sessão criou-se dois métodos, o primeiro é um método POST, o *startsession*, recurso 2 na Tab. (1), que tem como parâmetro da URL o módulo que se deseja acesso, além disso deve-se passar o certificado da aplicação cofdificado em Base64 para conferir se ele tem acesso ao *Device Server*, o modelo de objeto JSON que deve ser enviado

Tabela 1 – Tabela de recursos

	Método	URL
1	GET	http://localhost:9977/deviceserver/
2	POST	http://localhost:9977/deviceserver/startsession /{module}
3	POST	http://localhost:9977/deviceserver/establishsession /{session}
4	POST	http://localhost:9977/deviceserver/{session}

a esse método é o seguinte:

```
{
    certificate: "certificate"
}
```

Chegando ao *Device Server* ele checa se o módulo desejado existe e está carregado, depois verifica a validade do certificado enviado junto a requisição de inicialização da sessão. Tudo estando correto ele gera os parâmetros do protocolo de Diffie-Hellman e responde à aplicação com o seguinte modelo de objeto JSON:

```
{
    p: "Parametro P do Diffie-Hellman",
    g: "Parametro G do Diffie-Hellman",
    y: "Parametro Y do Diffie-Hellman",
    session: "Identidicador da sessao"
}
```

A aplicação em posse dessa resposta gera a sua parte dos parâmetros do protocolo Diffie-Hellman, assina-os com a chave correspondente ao seu certificado e envia os parâmetros e a assinatura para o segundo método para o estabelecimento da sessão, o *establishsession*, recurso 3 da Tab. (1), contendo o modelo de objeto JSON a seguir:

```
{
    p: "Parametro P do Diffie-Hellman",
    g: "Parametro G do Diffie-Hellman",
    y: "Parametro Y do Diffie-Hellman",
    signature: "Assinatura dos parametros acima"
```

```
}
```

Agora o *Device Server* pode verificar a assinatura dos parâmetros e assim, se estive correta, autorizar a inicialização da sessão. A partir daí a aplicação pode se comunicar com o módulo através do método 4 da Tab. (1), o modelo de objeto JSON para comunicação é o seguinte:

```
{

    method: "Metodo do modulo cifrado",
    message: "Mensagem para o modulo cifrada",
    iv: "Initialization vector da cifragem dos parametros"
}
```

O atributo *method* e *message* são crifrados com a chave de sessão criados no início da sessão, e o atributo *iv* contém o vetor de inicialização da cifragem. O atributo *message* pode conter qualquer string que o módulo definir, Ex. base64, JSON, texto plano etc.

4.1.1 Módulos

O *Device Server* foi pensado para ser o meio para que aplicações web e dispositivos se comuniquem com segurança e facilidade, pensando nisso resolveu-se não amarrar nenhuma implementação de acesso a dispositivo com o *Device Server* em si, resolvemos usar a abordagem de programação orientada a componentes, onde os módulos possam ser encaixados no *Device Server* e assim serem acessados com a intermediação dele, o diagrama de componentes é mostrado na Fig. (12).

A definição da interface a ser seguida pelos módulos é a seguinte:

```
package br.com.trapp.deviceserver.api;

public interface DeviceModule {

    public DeviceMessage incomingMessage(DeviceMessage
        message) throws DeviceException;

    public String getName();

    public String getVersion();

    public String getIdendifier();
```

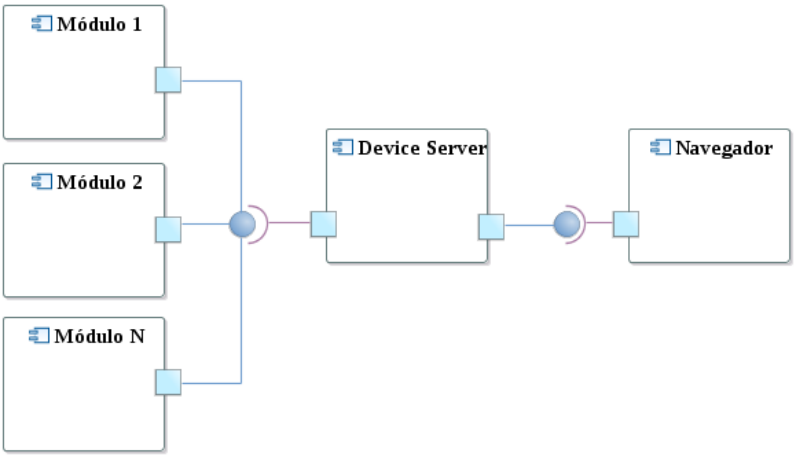

Figura 12 – Diagrama de componentes.

```
}
```

Com isso temos um ponto de comunicação que sempre deverá ser seguido quando implementado um novo módulo. O método *incomingMessage* é o método responsável por receber mensagem vindas do *Device Server*, e o retorno desse método é a respota que o *Device Server* dará a aplicação que fez a requisição.

O carregamento dos módulos é feito na inicialização do *Device Server*, que procura dentro da pasta onde foi executado o diretório *modules* e carrega todos os arquivos ".jar"que estão nessa pasta e respeitam a interface previamente definida.

4.2 ECHO MODULE

Para testarmos a solução foi implementado um módulo simples para o *Device Server*, ele possui dois métodos, o método *echo* que retorna a mesma mensagem que a aplicação lhe enviou e o método *echo-reverse* que retorna o reverso da mensagem enviada a ele. Além disso desenvolvemos uma pagina em HTML (Fig. (13)) e alguns scripts em Javascript para se comunicar com o *Device Server* e verificar o correto

funcionamento. Num cenário real o uso do *Device Server* por uma aplicativo Javascript é desaconselhável por expõe a chave privada do certificado da aplicação.

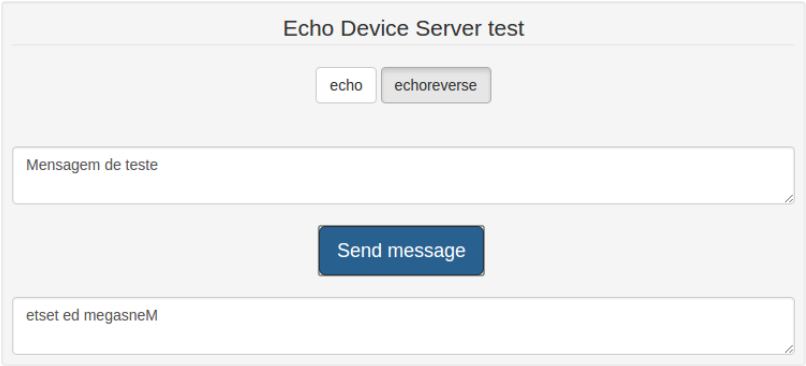

Figura 13 – Página da aplicação de teste.

Com a aplicação de teste além da verificação do correto funcionamento do *Device Server*, medimos os tempos para a estabilização da sessão e das consecutivas comunicações afim de verificar qual é o custo do uso dessa solução. Como o módulo *echo* é extremamente simples desconsiderou-se o tempo de processamento dele pois é irrisório comparado com o tempo do processamento da requisição, o medição foi feita executando cada teste cinco vezes para tirarmos uma média. na Tab. (2) podemos ver os tempos (em milissegundos) que cada repetição levou para estabelecer uma sessão com o módulo. E na Tab. (3) os tempos entre o envio e o recebimento de uma mensagem ao módulo.

Tabela 2 – Tempos para estabelecimento de sessão

Experimento	Tempo
1	718 ms
2	870 ms
3	903 ms
4	881 ms
5	872 ms
Média	848,8 ms

Tabela 3 – Tempos de comunicação

Experimento	Tempo
1	11 ms
2	10 ms
3	8 ms
4	10 ms
5	9 ms
Média	9,6 ms

4.3 CRYPTOGRAPHIC MODULE

Visando um resultado mais concreto também desenvolvemos um módulo capaz de se comunicar com dispositivos criptográficos. Esse módulo consegue listar dispositivos compatíveis, listar certificados e realizar assinatura digital. Os métodos implementados são:

Tabela 4 – Métodos Cryptographic Module

Método	Descrição
list	Lista os dispositivos compativeis conectados
listCerts	Lista os certificados dos dispositivos escolhidos
sign	Assina uma mensagem com a chave escolhida

Para testar esse módulo desenvolvemos também uma nova aplicação web desenvolvida com Javascript e HTML (Fig. (14)).

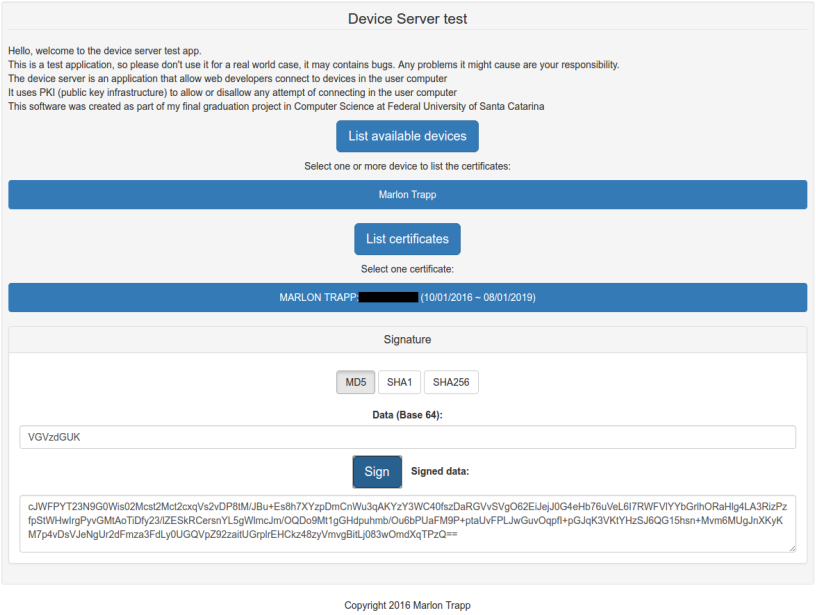

Figura 14 – Página da aplicação de teste do Cryptographic Module.

5 CONCLUSÃO

Nesse trabalho foi apresentado o conceito de aplicações web e sua necessidade de, em alguns casos, ter acesso a dispositivos conectados no computador do usuário. Uma maneira de se acessar dispositivos através do navegador é o uso do plugin Java Applet. Porém, o navegador mais utilizado atualmente desativou o suporte a esse plugin. Esse trabalho propôs uma alternativa a esse plugin, chamado de *Device Server*.

No capítulo 3 mostrou-se como é possível desacoplar o *Device Server* de um navegador específico usando requisições HTML e Javascript, duas tecnologias presentes em todos os navegadores modernos. Propomos ainda, a independência de plataforma utilizando a linguagem de programação Java, a qual pode ser executada na maioria dos SOs atuais.

Propomos também, para resolver o problema de acesso a diferentes tipos de dispositivos, usar uma abordagem modular. Cada módulo é dedicado a uma classe ou tipo de dispositivo. Isso deixou a solução mais genérica permitindo o desenvolvimento posterior de módulos para dispositivos não planejados inicialmente.

Na seção 3.3 mostrou-se como a utilização de certificados digitais pode ser usada para autenticar a aplicação que deseja acesso à nossa solução. Isso garante o acesso somente para quem deveria poder acessar os dispositivos. Ainda na questão de segurança, alcançou-se o sigilo da comunicação entre o *Device Server* e a aplicação web através o protocolo TLS, o qual no protótipo foi simplificado e utilizado a troca de chaves de Diffie Hellman. Esses dois artefatos nos garantem autenticidade e sigilo, aumentando a segurança da solução.

No capítulo 4 desenvolvemos uma prova de conceito do modelo proposto implementando todas as funcionalidades propostas no capítulo 3. Ainda nesse capítulo, na seção 4.2, desenvolvemos um módulo para acoplarmos ao *Device Server* e testar seu funcionamento. O qual nos deu um resultado preliminar quanto à performance da solução proposta. Em sequencia, na seção 4.3, foi apresentado outro módulo que faz a comunicação com dispositivos criptográficos reais conectados no computador do usuário.

5.1 TRABALHOS FUTUROS

Através desse trabalho o objetivo de criar uma alternativa ao Java Applet para acesso a dispositivos pelo navegador foi alcançada. Porém essa solução inicial deve ser mais trabalhada, uma das principais melhorias que poderiam acontecer seria a implementação da comunicação do navegador e o *Device Server* através de *websockets*, isso permitiria a comunicação assíncrona, o que não é possível nessa versão.

Outro trabalho futuro necessário é a implantação do protocolo TLS no protótipo. Atualmente o protocolo TLS foi substituído pelo acordo de troca de chaves de *Diffie-Hellman* por questões de simplificação da implementação.

Finalmente, poderia-se executar mais testes com dispositivos que exigem mais banda para a comunicação e de ataques conhecidos para obter-se resultados quanto a performance e segurança da solução.

REFERÊNCIAS

ADAMS, C.; LLOYD, S. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. S.l.: Addison-Wesley, 2003. 322 p.

HOUSLEY, R.; POLK, T. *Planning for PKI: Best Practices Guide for Deploting Public Key Infrastructure*. S.l.: John Wiley & Sons, Inc, 2001. 327 p.

MENEZES, A. J.; OORSCHOT, P. C. van; VANSTONE, S. A. *Handbook of Applied Cryptography*. 5. ed. S.l.: CRC Press, 2001.

NETMARKETSHARE. *Desktop Browser Market Share*. 2016. <https://www.netmarketshare.com/browser-market-share.aspx?qprid=0qpcustomd=0qptimeframe=Mqpsp=207>. Acessado em 02/05/2016.

SCHNEIER, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. New York, N.Y.: John Wiley & Sons, Inc, 1996. 675 p.

SCHUH, J. *The Final Countdown for NPAPI*. 2014. <http://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html>. Acessado em 02/05/2016.

STALLINGS, W. *Cryptography and Network Security: Principles and Practices*. Upper Saddle River, N.J.: Pearson Education, Inc., 2003. 680 p.

TOPIC, D. *Moving to a Plugin-Free Web*. 2016. <https://blogs.oracle.com/java-platform-group/entry/moving_to_a_plugin_free>. Acessado em 03/05/2016.

ANEXO A – Código fonte Device Server API

```
package br.com.trapp.deviceserver.api;

public class DeviceException extends Exception {

  /**
     *
     */
  private static final long serialVersionUID = 1L;

  public DeviceException(String msg, Throwable throwable) {
    super(msg, throwable);
  }

  public DeviceException(String msg) {
    super(msg);
  }

}
```

```
package br.com.trapp.deviceserver.api;

import java.math.BigInteger;
import java.util.Random;

public class DH {

  private BigInteger y;
  private BigInteger x;
  private BigInteger p;
  private BigInteger k;
  private BigInteger g;

  public DH(BigInteger p, BigInteger g) {
    this.g = g;
    this.p = p;
    Random random = new Random();
    do {
      x = new BigInteger(p.bitCount(), random);
    } while (x.compareTo(p) >= 0);
    this.y = g.modPow(x, p);
  }

  public BigInteger getY() {
    return this.y;
  }

  public void setOtherY(BigInteger y2) {
    this.k = y2.modPow(x, p);
```

```
  }

  public BigInteger getSharedSecret() {
    return this.k;
  }

  public BigInteger getP() {
    return this.p;
  }

  public BigInteger getG() {
    return this.g;
  }
}
```

```
package br.com.trapp.deviceserver.api;

public class AuthorizationProccess {

  private String p;
  private String g;
  private String y;
  private String signature;
  private int session;

  public AuthorizationProccess() {
  }

  public AuthorizationProccess(String p, String g,
      String y, int session) {
    this.p = p;
    this.g = g;
    this.y = y;
    this.session = session;
  }

  public String getP() {
    return p;
  }

  public void setP(String p) {
    this.p = p;
  }

  public String getG() {
    return g;
  }

  public void setG(String g) {
    this.g = g;
  }
```

```
  public String getY() {
    return y;
  }

  public void setY(String y) {
    this.y = y;
  }

  public int getSession() {
    return session;
  }

  public void setSession(int session) {
    this.session = session;
  }

  public String getSignature() {
    return signature;
  }

  public void setSignature(String signature) {
    this.signature = signature;
  }
}
```

```
package br.com.trapp.deviceserver.api;

/**
 * Interface that every module have to follow in order to
 * connect it in the device server
 *
 * @author marlon
 *
 */
public interface DeviceModule {

  /**
   * This method is from where all messages will come to the
   * module
   *
   * @param message
   *          the message
   * @return the return message null if none
   */
  public DeviceMessage
      incomingMessage(DeviceMessage message)
          throws DeviceException;

  /**
   * This should return the beauty name of the module
   *
   * @return the version of the module
```

```
     */
    public String getName();

    /**
     * This should return the version of the module
     *
     * @return the version of the module
     */
    public String getVersion();

    /**
     * This should return the identifier to be used in the
     * path in the Device Server
     *
     * @return the name that will identify the module
     */
    public String getIdendifier();

}

package br.com.trapp.deviceserver.api;

public class Authorization {

    /**
     * Certificate encoded in PEM
     */
    private String certificate;

    public Authorization(String certificate) {
        this.certificate = certificate;
    }

    public Authorization() {
    }

    public String getCertificate() {
        return certificate;
    }

    public void setCertificate(String certificate) {
        this.certificate = certificate;
    }

}

package br.com.trapp.deviceserver.api;

/**
 * Class to encapsulate messages through the device server.
 * This class is generic enough to accept any type of
 * message needed by the module.
```

```
 * <p>
 * DO NOT USE THIS CLASS DIRECTLY IN SERVER. This class is
 * supposed to be used directly only in the module. To
 * construct a DeviceMessage use the
 * {@link DeviceMessageFactory} class.
 *
 * @author marlon
 *
 */
public class DeviceMessage {

  private String method;

  private String message;

  private String iv;

  /**
   * /** Default constructor
   *
   * @param method
   *          the method of the target module to be executed
   * @param message
   *          it can be anything defined in the module
   *          documentation. Eg. plain text or json
   */
  public DeviceMessage(String method, String message) {
    this.message = message;
  }

  /**
   * Empty constructor for serialization
   */
  public DeviceMessage() {
  }

  public String getMessage() {
    return this.message;
  }

  public void setMessage(String message) {
    this.message = message;
  }

  public String getMethod() {
    return method;
  }

  public void setMethod(String method) {
    this.method = method;
  }
```

```
  public void setIV(String iv) {
    this.iv = iv;
  }

  public String getIV() {
    return this.iv;
  }

}
```

ANEXO B – Código fonte Device Server

```
package br.com.trapp.deviceserver.model.session;

import java.io.ByteArrayInputStream;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SecureRandom;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;

import org.bouncycastle.crypto.CipherParameters;
import org.bouncycastle.crypto.InvalidCipherTextException;
import org.bouncycastle.crypto.engines.AESEngine;
import org.bouncycastle.crypto.modes.CBCBlockCipher;
import org.bouncycastle.crypto.paddings.BlockCipherPadding;
import org.bouncycastle.crypto.paddings.PKCS7Padding;
import org.bouncycastle.crypto.paddings.
PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.params.KeyParameter;
import org.bouncycastle.crypto.params.ParametersWithIV;
import org.bouncycastle.util.encoders.Base64;

import br.com.trapp.deviceserver.api.DH;
import br.com.trapp.deviceserver.api.DeviceException;
import br.com.trapp.deviceserver.api.DeviceMessage;
import br.com.trapp.deviceserver.api.DeviceModule;

public class Session {

  private Integer id;
  private DH dh;
  private DeviceModule module;
  private PaddedBufferedBlockCipher cipher;
  private byte[] keyBytes;
  private String certificate;
  private long lastUse = System.currentTimeMillis();

  public Session(Integer id, DH dh, DeviceModule module,
      String certificate) {
    this.id = id;
    this.dh = dh;
    this.module = module;
    this.certificate = certificate;
  }

  public DH getDh() {
    return dh;
  }
```

```
public void setDh(DH dh) {
  this.dh = dh;
}

public DeviceModule getModule() {
  return module;
}

public void setModule(DeviceModule module) {
  this.module = module;
}

public String getCertificate() {
  return certificate;
}

public void setCertificate(String certificate) {
  this.certificate = certificate;
}

public void init() throws NoSuchAlgorithmException,
    NoSuchProviderException {
  MessageDigest digest =
      MessageDigest.getInstance("SHA256", "BC");
  byte[] keyByteArray =
      dh.getSharedSecret().toString().getBytes();
  digest.update(keyByteArray);
  this.keyBytes = digest.digest();

  // setup AES cipher in CBC mode with PKCS7 padding
  BlockCipherPadding padding = new PKCS7Padding();
  this.cipher =
      new PaddedBufferedBlockCipher(new CBCBlockCipher(
          new AESEngine()), padding);
}

/**
 * Encrypt the given message
 *
 * @param message
 *          the message to be encrypted
 *
 * @return the message encrypted
 *
 * @throws InvalidCipherTextException
 */
public DeviceMessage encrypt(DeviceMessage message)
    throws InvalidCipherTextException {

  SecureRandom secureRandom = new SecureRandom();
  byte[] iv = new byte[16];
```

```
  secureRandom.nextBytes(iv);
  message.setIV(Base64.toBase64String(iv));
  return processCipher(message, true);
}

/**
 * Decrypt the given message
 *
 * @param message
 *          the message to be decrypted
 *
 * @return the message decrypted
 *
 * @throws InvalidCipherTextException
 */
public DeviceMessage decrypt(DeviceMessage message)
    throws InvalidCipherTextException {
  return this.processCipher(message, false);
}

/**
 * Process the message, encrypting or decrypting it
 * depends on crypt parameter
 *
 * @param message
 *          the message to be processed
 * @param crypt
 *          desirable operation, encrypt or decrypt
 *
 * @return the message encrypted or decrypted
 *
 * @throws InvalidCipherTextException
 */
private DeviceMessage processCipher(
    DeviceMessage message, boolean crypt)
    throws InvalidCipherTextException {
  cipher.reset();
  KeyParameter keyParam = new KeyParameter(this.keyBytes);
  CipherParameters params =
      new ParametersWithIV(keyParam,
          Base64.decode(message.getIV()));
  cipher.init(crypt, params);

  String method = null;
  String msg = null;
  if (crypt) {
    if (message.getMethod() != null)
      method =
          Base64.toBase64String(this.process(message
              .getMethod().getBytes()));
    if (message.getMessage() != null)
      msg =
```

```
            Base64.toBase64String(this.process(message
                .getMessage().getBytes()));
    } else {
      // We have to add .trim() because of the strange
      // behavior of the
      // string
      if (message.getMethod() != null)
        method =
            new String(this.process(Base64.decode(message
                .getMethod()))).trim();
      if (message.getMessage() != null)
        msg =
            new String(this.process(Base64.decode(message
                .getMessage()))).trim();
    }
    message.setMethod(method);
    message.setMessage(msg);
    return message;
  }

  /**
   * Do the encrypt or decrypt operation depends on the
   * initialization of the cipher
   *
   * @param data
   *          The data to be processed
   * @return the processed data
   *
   * @throws InvalidCipherTextException
   */
  private byte[] process(byte[] data)
      throws InvalidCipherTextException {
    byte[] buf =
        new byte[cipher.getOutputSize(data.length)];
    int len =
        cipher.processBytes(data, 0, data.length, buf, 0);
    len += cipher.doFinal(buf, len);
    return buf;
  }

  public DeviceMessage sendMessageToModule(
      DeviceMessage message) throws DeviceException,
      InvalidCipherTextException {
    DeviceMessage decryptedMessage = this.decrypt(message);
    DeviceMessage response =
        module.incomingMessage(decryptedMessage);
    DeviceMessage encryptedResponse =
        this.encrypt(response);
    lastUse = System.currentTimeMillis();
    return encryptedResponse;
  }
```

```
  public Certificate generateCertificate()
      throws CertificateException {

    ByteArrayInputStream byteArrIStream =
        new ByteArrayInputStream(certificate.getBytes());
    CertificateFactory cf =
        CertificateFactory.getInstance("X.509");
    return cf.generateCertificate(byteArrIStream);
  }

  @Override
  public String toString() {
    return module.getName() + " - " + id;
  }

  public long getLastUse() {
    return lastUse;
  }

  public Integer getId() {
    return id;
  }

  public void setId(Integer id) {
    this.id = id;
  }

}

package br.com.trapp.deviceserver.model;

import java.math.BigInteger;
import java.util.Random;

public class DH {

  private BigInteger y;
  private BigInteger x;
  private BigInteger p;
  private BigInteger k;
  private BigInteger g;

  public DH(BigInteger p, BigInteger g) {
    this.g = g;
    this.p = p;
    Random random = new Random();
    do {
      x = new BigInteger(p.bitCount(), random);
    } while (x.compareTo(p) >= 0);
    this.y = g.modPow(x, p);
  }
```

```
  public BigInteger getY() {
    return this.y;
  }

  public void setOtherY(BigInteger y2) {
    this.k = y2.modPow(x, p);
  }

  public BigInteger getSharedSecret() {
    return this.k;
  }

  public BigInteger getP() {
    return this.p;
  }

  public BigInteger getG() {
    return this.g;
  }
}
```

```
package br.com.trapp.deviceserver.model;

import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.PublicKey;
import java.security.Signature;
import java.security.SignatureException;
import java.security.cert.CertificateException;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.Set;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import org.bouncycastle.util.encoders.Base64;

import br.com.trapp.deviceserver.api.Authorization;
import br.com.trapp.deviceserver.api.AuthorizationProccess;
import br.com.trapp.deviceserver.api.DH;
import br.com.trapp.deviceserver.api.DeviceMessage;
import br.com.trapp.deviceserver.model.exception.
AuthorizationException;
import br.com.trapp.deviceserver.model.exception.
ModuleManagerException;
import br.com.trapp.deviceserver.model.module.ModuleManager;
import br.com.trapp.deviceserver.model.session.Session;
import br.com.trapp.deviceserver.utils.PKIXUtils;
```

```
public class DeviceBridge {

  public static BigInteger P = new BigInteger(
      "32055214315510634381537827598970291171729459495 1"
          + "92300976534889158245046312148624568368116 1"
          + "74970060200188393185483894637311419069066 9"
          + "72454117106429810600084203078743722566218 7"
          + "04858644015095089755745030120640858760887 1"
          + "85150813126633789416744251610542498968893 6"
          + "31451485209041100318598384609705087112758 0"
          + "12553170392433014961298894638708544655380 2"
          + "57495045752696579411864393447647498181598 4"
          + "78624718482784674086185691451669563469339 6"
          + "02058683840298648620398595032810587183079 8"
          + "05104421627779043329674998418624800701832 4"
          + "88500009594242667639002071155719251569396 7"
          + "23263497716700430440410574382097997040868 5"
          + "19633997621281861142699");
  public static BigInteger G = new BigInteger("2");
  public static int SESSION_TIMEOUT = 15;

  public Map<Integer, Session> authorizedSessions;
  public Map<Integer, Session> inAuthorizationSessions;

  private final ScheduledExecutorService scheduler;

  /**
   * Initialize the device bridge with a cleaner thread to
   * clean old sessions
   */
  public DeviceBridge() {
    authorizedSessions = new HashMap<Integer, Session>();
    inAuthorizationSessions =
        new HashMap<Integer, Session>();
    scheduler = Executors.newScheduledThreadPool(1);
    Runnable cleaner = new Runnable() {
      @Override
      public void run() {
        long current = System.currentTimeMillis();
        Set<Integer> keys = authorizedSessions.keySet();
        for (Integer integer : keys) {
          long lastUse =
              authorizedSessions.get(integer).getLastUse();
          if (lastUse < current
              - (1000 * 60 * SESSION_TIMEOUT)) {
            authorizedSessions.remove(integer);
          }
        }
        keys = inAuthorizationSessions.keySet();
        for (Integer integer : keys) {
          long lastUse =
```

```
            inAuthorizationSessions.get(integer)
                .getLastUse();
        if (lastUse < current
            - (1000 * 60 * SESSION_TIMEOUT)) {
          inAuthorizationSessions.remove(integer);
        }
      }
    }
  };
  scheduler.scheduleAtFixedRate(cleaner, 15, 15,
      TimeUnit.SECONDS);
}

/**
 * Start session method, it starts the authorization of
 * access to a module
 *
 * @param module
 *          The target module to authorize
 * @param auth
 *          Authorization parameter
 * @return the authorization process with information to
 *         agree in an shared key
 *
 * @throws CertificateException
 *           if the certificate is not valid
 * @throws AuthorizationException
 *           if the module is not loaded
 * @throws ModuleManagerException
 *           if the module
 */
public AuthorizationProccess startSession(String module,
    Authorization auth) throws CertificateException,
    AuthorizationException, ModuleManagerException {

  if (!ModuleManager.isModuleAvailable(module)) {
    throw AuthorizationException.NOT_LOADED;
  }
  PKIXUtils pkix = new PKIXUtils();
  pkix.verifyCertificate(auth.getCertificate());

  DH dh = new DH(DeviceBridge.P, DeviceBridge.G);
  String p = DeviceBridge.P.toString();
  String g = DeviceBridge.G.toString();
  String y = dh.getY().toString();
  Random rnd = new Random(dh.getY().longValue());
  int session = rnd.nextInt();
  if (session < 0)
    session *= -1;
  AuthorizationProccess authProccess =
      new AuthorizationProccess(p, g, y, session);
  inAuthorizationSessions.put(
```

```
      session,
      new Session(session, dh, ModuleManager
          .getInstanceOfModule(module), auth
          .getCertificate()));
  return authProccess;
}

public DeviceMessage sendMessage(Integer session,
    DeviceMessage message) throws AuthorizationException {
  if (authorizedSessions.containsKey(session)) {
    try {
      return authorizedSessions.get(session)
          .sendMessageToModule(message);
    } catch (Exception e) {
      throw new AuthorizationException(e.getMessage(), e);
    }
  } else {
    throw new AuthorizationException(
        "Session not initialized, please use the proper "
            + "method first");
  }
}

/**
 * Establish a session that is in process of authorization
 *
 * @param sessionId
 *          the session ID to establish
 * @param auth
 *          the authentication parameters
 * @return the session id if the session was correctly
 *         authenticated
 *
 * @throws AuthorizationException
 *           if the session wasn't authenticate
 */
public Integer establishSession(Integer sessionId,
    AuthorizationProccess auth)
    throws AuthorizationException {
  if (inAuthorizationSessions.containsKey(sessionId)) {
    try {
      Session session =
          inAuthorizationSessions.get(sessionId);
      if (this.checkSignature(auth, session)) {
        session.getDh().setOtherY(
            new BigInteger(auth.getY()));
        inAuthorizationSessions.remove(session);
        session.init();
        authorizedSessions.put(sessionId, session);
        return sessionId;
      } else {
        throw new AuthorizationException(
```

```
            "Session not initialized, the signature of "
                + "parameters do not match");
      }
    } catch (Exception e) {
      throw new AuthorizationException(e.getMessage(), e);
    }
  } else {
    throw new AuthorizationException(
        "Session not initialized, please use the proper "
            + "method first");
  }
}

/**
 * Checks if the parameters were signed by the certificate
 * holder
 *
 * @param auth
 *          authentication parameters
 * @param session
 *          session parameters
 * @return true if the signature is valid
 * @throws Exception
 *           if the signature is not valid
 */
private boolean checkSignature(
    AuthorizationProccess auth, Session session)
    throws Exception {

  PublicKey pubKey;
  try {
    pubKey = session.generateCertificate().getPublicKey();
    Signature sig =
        Signature.getInstance("SHA256withRSA");
    byte[] p =
        session.getDh().getP().toString().getBytes();
    byte[] g =
        session.getDh().getG().toString().getBytes();
    byte[] y = auth.getY().toString().getBytes();
    byte[] id = session.getId().toString().getBytes();
    sig.initVerify(pubKey);
    sig.update(id);
    sig.update(p);
    sig.update(g);
    sig.update(y);
    return sig.verify(Base64.decode(auth.getSignature()));
  } catch (CertificateException e) {
    throw new AuthorizationException(
        "The certificate is not corrected encoded", e);
  } catch (NoSuchAlgorithmException e) {
    throw e;
  } catch (InvalidKeyException e) {
```

```
      throw e;
    } catch (SignatureException e) {
      throw e;
    }
  }

  public Collection<Session> getSessions() {
    return authorizedSessions.values();
  }

}

package br.com.trapp.deviceserver.model.module;

public class ModuleID {

  private String name;
  private String identifier;

  public ModuleID() {
    // TODO Auto-generated constructor stub
  }

  public ModuleID(String name, String identifier) {
    this.name = name;
    this.identifier = identifier;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getIdentifier() {
    return identifier;
  }

  public void setIdentifier(String identifier) {
    this.identifier = identifier;
  }

  @Override
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result =
        prime
            * result
            + ((identifier == null) ? 0 : identifier
```

```
                .hashCode());
    result =
        prime * result
            + ((name == null) ? 0 : name.hashCode());
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    ModuleID other = (ModuleID) obj;
    if (identifier == null) {
      if (other.identifier != null)
        return false;
    } else if (!identifier.equals(other.identifier))
      return false;
    if (name == null) {
      if (other.name != null)
        return false;
    } else if (!name.equals(other.name))
      return false;
    return true;
  }

}

package br.com.trapp.deviceserver.model.module;

public class ModuleInfo {

  private String name;
  private String idendifier;
  private String version;

  public ModuleInfo(String name, String idendifier,
      String version) {
    this.name = name;
    this.idendifier = idendifier;
    this.version = version;
  }

  public ModuleInfo() {
  }

  public String getName() {
    return name;
  }
```

```
  public void setName(String name) {
    this.name = name;
  }

  public String getIdendifier() {
    return idendifier;
  }

  public void setIdendifier(String idendifier) {
    this.idendifier = idendifier;
  }

  public String getVersion() {
    return version;
  }

  public void setVersion(String version) {
    this.version = version;
  }

  @Override
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result =
        prime
            * result
            + ((idendifier == null) ? 0 : idendifier
                .hashCode());
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    ModuleInfo other = (ModuleInfo) obj;
    if (idendifier == null) {
      if (other.idendifier != null)
        return false;
    } else if (!idendifier.equals(other.idendifier))
      return false;
    return true;
  }

}
```

```
package br.com.trapp.deviceserver.model.module;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.lang.reflect.InvocationTargetException;
import java.net.URL;
import java.net.URLClassLoader;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.Vector;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;
import java.util.jar.Manifest;

import br.com.trapp.deviceserver.api.DeviceModule;
import br.com.trapp.deviceserver.model.TrustedCertificates;
import br.com.trapp.deviceserver.model.exception.
ModuleManagerException;

public class ModuleManager {

  private static Map<ModuleInfo, Class<?>> availableModules =
      new HashMap<ModuleInfo, Class<?>>();

  public static void loadModule(File file)
      throws ModuleManagerException {
    Class<?> module;
    try {
      module =
          ModuleManager.verifyJarSignature(file,
              (X509Certificate) TrustedCertificates
                  .getDevCert());
    } catch (CertificateException | IOException e) {
      throw new ModuleManagerException(
          e.getLocalizedMessage(), e);
    }
    if (module != null) {
      availableModules
          .put(ModuleManager.getModuleIdentifier(module),
              module);
    } else {
      throw ModuleManagerException.NOT_A_MODULE;
    }
```

```
}

/**
 * Compare it to the expected X509Certificate. If
 * everything went well and the certificates are the same,
 * no exception is thrown.
 */
private static Class<?> verifyJarSignature(File file,
    X509Certificate targetCert) throws IOException {

  ClassLoader cl =
      URLClassLoader.newInstance(new URL[] { new URL(
          "jar:file:" + file.getAbsolutePath() + "!/") });
  JarFile jarFile = new JarFile(file);
  Class<?> portClass = null;
  // Sanity checking
  if (targetCert == null) {
    jarFile.close();
    throw new SecurityException(
        "Provider certificate is invalid");
  }

  Vector<JarEntry> entriesVec = new Vector<JarEntry>();

  // Ensure the jar has a manifest file.
  Manifest man = jarFile.getManifest();
  if (man == null) {
    jarFile.close();
    throw new SecurityException(
        "The jar is not signed, missing manifest file");
  }

  // Ensure all the entries' signatures verify
  // correctly
  byte[] buffer = new byte[8192];
  Enumeration<JarEntry> entries = jarFile.entries();

  while (entries.hasMoreElements()) {
    JarEntry je = (JarEntry) entries.nextElement();

    // Skip directories.
    if (je.isDirectory())
      continue;
    entriesVec.addElement(je);
    InputStream is = jarFile.getInputStream(je);

    // Read in each jar entry. A security exception
    // will
    // be thrown if a signature/digest check fails.
    while (is.read(buffer, 0, buffer.length) != -1) {
      // Don't care
    }
```

```
    is.close();
  }

  // Get the list of signer certificates
  Enumeration<JarEntry> e = entriesVec.elements();

  while (e.hasMoreElements()) {

    JarEntry je = e.nextElement();
    try {

      if (je.getName().endsWith(".class")) {
        // -6 because of .class
        String className =
            je.getName().substring(0,
                je.getName().length() - 6);
        className = className.replace('/', '.');
        Class<?> clazz;
        clazz = cl.loadClass(className);

        if (Arrays.stream(clazz.getInterfaces())
            .anyMatch(
                claz -> claz.equals(DeviceModule.class))) {
          portClass = clazz;
        }
      }
    } catch (ClassNotFoundException e1) {
      // Ignore
    }
    // Every file must be signed except files in
    // META-INF.
    Certificate[] certs = je.getCertificates();
    if ((certs == null) || (certs.length == 0)) {
      if (!je.getName().startsWith("META-INF")) {
        jarFile.close();
        throw new SecurityException("The jar "
            + "has unsigned " + "class files.");
      }
    } else {
      // Check whether the file is signed by the
      // expected
      // signer. The jar may be signed by multiple
      // signers.
      // See if one of the signers is
      // 'targetCert'.
      boolean signedAsExpected = false;

      for (int i = 0; i < certs.length; i++) {
        if (certs[i].equals(targetCert)) {
          // Stop since one trusted signer is
          // found.
          signedAsExpected = true;
```

```
            break;
          }
        }

        if (!signedAsExpected) {
          jarFile.close();
          throw new SecurityException("The jar "
              + "is not signed by a " + "trusted signer");
        }
      }
    }
    jarFile.close();
    return portClass;
  }

  private static ModuleInfo getModuleIdentifier(
      Class<?> module) throws ModuleManagerException {
    try {
      DeviceModule inst =
          ((DeviceModule) module.getConstructor()
              .newInstance());
      return new ModuleInfo(inst.getName(),
          inst.getIdendifier(), inst.getVersion());
    } catch (InstantiationException
        | IllegalAccessException | IllegalArgumentException
        | InvocationTargetException | NoSuchMethodException
        | SecurityException e) {
      throw new ModuleManagerException(e.getMessage(), e);
    }
  }

  public static DeviceModule getInstanceOfModule(
      String identifier) throws ModuleManagerException {
    try {
      ModuleInfo moduleInfo = new ModuleInfo();
      moduleInfo.setIdendifier(identifier);
      Class<?> clazz =
          ModuleManager.availableModules.get(moduleInfo);
      DeviceModule moduleInstance =
          ((DeviceModule) clazz.getConstructor()
              .newInstance());
      return moduleInstance;
    } catch (InstantiationException
        | IllegalAccessException | IllegalArgumentException
        | InvocationTargetException | NoSuchMethodException
        | SecurityException e) {
      throw new ModuleManagerException(e.getMessage(), e);
    }
  }

  /**
   * Get the loaded modules info
```

```
     *
     * @return a set with the info of the loaded modules
     */
    public static Set<ModuleInfo> getAvailableModules() {
      return availableModules.keySet();
    }

    public static boolean isModuleAvailable(String module) {
      return availableModules.keySet().parallelStream()
          .anyMatch(k -> k.getIdendifier().equals(module));
    }

    public static List<String> getAvailableModulesNames()
        throws ModuleManagerException {
      Set<ModuleInfo> keys = availableModules.keySet();
      List<String> result = new ArrayList<String>();
      for (ModuleInfo name : keys) {
        result.add(name.getName());
      }
      return result;
    }
}
```

```
package br.com.trapp.deviceserver.model;

import java.io.ByteArrayInputStream;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;

public class TrustedCertificates {

  public static final String CA_CERT =
      "-----BEGIN CERTIFICATE-----\n"
          + "MIIKFTCCBf2gAwIBAgIJAPkHix2mWttkMA0GCSqGSIb3DQE"
          + "BCwUAMH8xCzAJBgNV\n"
          + "BAYTAkJSMRcwFQYDVQQIDA5TYW50YSBDYXRhcmluYTEWMBQ"
          + "GA1UEBwwNRmxvcmlh\n"
          + "bm9wb2xpczEOMAwGA1UECgwFVHJhcHAxFDASBgNVBAsMC0R"
          + "ldmVsb3BtZW50MRkw\n"
          + "FwYDVQQDDBBEZXZpY2UgU2VydmVyIENBMB4XDTE2MDMwMTI"
          + "yMjUyNFoXDTI2MDIy\n"
          + "NzIyMjUyNFowfzELMAkGA1UEBhMCQlIxFzAVBgNVBAgMDlN"
          + "hbnRhIENhdGFyaW5h\n"
          + "MRYwFAYDVQQHDA1GbG9yaWFub3BvbGlzMQ4wDAYDVQQKDAV"
          + "UcmFwcDEUMBIGA1UE\n"
          + "CwwLRGV2ZWxvcG1lbnQxGTAXBgNVBAMMEERldmljZSBTZXJ"
          + "2ZXIgQ0EwggQiMA0G\n"
          + "CSqGSIb3DQEBAQUAA4IEDwAwggQKAoIEAQDLerSTWUEWUgR"
          + "n0uaLpPn9la7YIzgI\n"
          + "qxnObaq7c+mZ9/n3lwbase99RAMjwmASzvshQhOhvk1QbWy"
          + "E5tIqsB2592GmFCrY\n"
```

```
+ "NdNfQnfA7+87BoHGhm/Y/eQCXVAj87merQPirqOKE3gDw77"
+ "2eu92oLxRXvI8OJin\n"
+ "rEwuHpL/N8j5TvXViAHu1XuYVm4vHGRVuR9X8dAzw5hWYeY"
+ "Wiq/gPACM54Oao1oG\n"
+ "WTVIeyrZKjeZzgRPERNlpRO1sXJhNUkj+R5pJs0NIYZu3S4"
+ "A4XeD2FrOFE/TS84S\n"
+ "zwQEIDwPFMl7JwRoMOHzBczxwjsa05EMJHu2OQgROFEdXZ8"
+ "Srpop3wBuq5q6wicZ\n"
+ "ntzaQRmaXhyx04UCcfeyS09QYtmWEnv6BVF4JRPm5MB/kFi"
+ "HVr3Y0J9av981JKzj\n"
+ "Xw9G3nio/Rkd3PWYyOabhpOIPdpPBQb1Etd8gxLmjlsrxf/"
+ "Cc48oPQavDA0ZpNeH\n"
+ "tUsx6avkQphOMYFsIW/vh5EbQAs0aVzn7jKzTR2YqvUNZD5"
+ "AbK0Ovn/5xNH0JqbP\n"
+ "m5WhF15cUvnE/2nmbYVKoXvVSgkrPEQOwCLc+1TJ8y4SBWX"
+ "tIat5y0aLCUXsVqdJ\n"
+ "ZsEoFAJ95dLWmwvky88j/p9Z2e6o+XyS1SfUcbtiMGbEoZx"
+ "EzBlnEyB6J2yNTx5e\n"
+ "NvUUPf5Fy4KBiRo8V0+rfqIgG39+vOz7u9fJFnRM6laf3Mf"
+ "HXSzqmHINrXuGHH/e\n"
+ "rPs6zw2fFnD8Lrza7rQleTo8hSnOBjwBGbwI+9RyqmD7zQd"
+ "fEwQp40NoQl0HzmMo\n"
+ "V/hCseqr1GeEHYEofN9H7Dv0gZFV2vI9FGKKYZdZVctf4ac"
+ "/oB7uUlfS+FLw8a0k\n"
+ "y6q+VEbHAs8OFHR5Q+ZXQZO83mxAVI86c2KabQiNu4iGKj1"
+ "0o0KYFDEgTmYADWxa\n"
+ "DrNOdqtaZu03upn1dEpp5AFqMp/Fk1ctA8BW30r6qaXYkTb"
+ "iOlU7FOP3hUL90IVg\n"
+ "/e/Of7kA12MEAyJslh6vFpm1hegCKPiP0qzvPVszfk0chtu"
+ "y3cqC+EKIwf6ijDyo\n"
+ "sF3IaW+FcdFtoStI3npGZWDYpTq/b9NhfeJh2q8+Vi8Yrd8"
+ "KDCkINkHLsnq5GUGG\n"
+ "9HD96o2N3WA7NaaV7PiKxqy/BjZO7mlPbjXmtCobTpq573X"
+ "LkiDWRCAFWukaax3F\n"
+ "4sITnqjmLLyLdzY2hnPXyahnuIHynACjgAuUxDMA9YedzSg"
+ "TIGXodsMBzsAHp7Fa\n"
+ "MDu6ars1fx2ogrJaYgI2iOMtmw5Fkv22oHilJe4k42RQDQe"
+ "9Yi1ezmX3bhbjJ2xu\n"
+ "SlsgCTeYUL9kEfxDsnN9anaGOBdJ1KoYIx06g6IRblcJM7R"
+ "kSB4tvf5XAgMBAAGj\n"
+ "gZMwgZAwHQYDVR0OBBYEFKnucAgamf7uE8Gl9E1zH07SucI"
+ "VMB8GA1UdIwQYMBaA\n"
+ "FKnucAgamf7uE8Gl9E1zH07SucIVMAwGA1UdEwQFMAMBAf8"
+ "wQAYDVR0fBDkwNzA1\n"
+ "oDOgMYYvaHR0cDovL3d3dy5pbmYudWZzYy5ici9+bWFybG9"
+ "uLnRyYXBwL2NhLWNy\n"
+ "bC5wZW0wDQYJKoZIhvcNAQELBQADggQBADJC7BMS/r5/Wvn"
+ "xbRF5y9hKhFaQGioz\n"
+ "FxoLXIFWeII3LC2V/by8K9KM2Z122ypHP7yi7gxlpL6rWx2"
+ "gl4Ovp1xP82BQyEMd\n"
+ "ac5U8ocU6Mk3aa4GsrCt764Ukm/IeMqWlBrCmhoS/UwVRwC"
+ "tDBZzaHzRDasw86Gm\n"
```

```
            + "WEgcsNZXALOJZe1WByHy/koRz6Q+oe+lg4IFYtj90kZhSB1"
            + "HcklF7l4lFOFrnNZH\n"
            + "Rk5UEGJor9Exxu2fFGX8/GzHsYwogAIqg1zWj0vsI1OQ09A"
            + "X/lNCN4EOSYa9GJYM\n"
            + "IGh4JZ6gatgDL9uofC5fx1Q+EMGw3Xtn6IUvR6T6rKBH5at"
            + "PzXBxTSc4nmOE1xzM\n"
            + "7Ut7hfkINnOCDJpa/HKpmmmgZVF994AQrLlI3WSgfd/UQU4"
            + "BT1PrfapqFcntUzb1\n"
            + "JM1LKAUZg4c25DxGnQI3UKs8L76AQ0x04HAvb1q3i6b8QcL"
            + "qtJHNKzXd4FSXC032\n"
            + "sFFqZdM7boNIwouqVAV964YVE01RWorF2qRV+pJu+TKfa9y"
            + "WG41LDy4oay3ZCasm\n"
            + "2Rihal5QzlCdOP5rSCU3zYEmrJR7MSsnXSJlIAhrUuHTh1t"
            + "23aVrVJa6t7H9Dx3K\n"
            + "6ki7SwlQU81qjVrxqDaBE9B/Bd/0ZbhM6hkQ7x+WHgVt5vi"
            + "isYoHHNbcz6XmoZMG\n"
            + "+PH5eMxUMf/taUv0txE3ySwu06kmFLfakisplXEO2Dr1erK"
            + "tCei5WYLOBMQZyQ0m\n"
            + "qwj9ne0n4eIC/1f3/5rN1rLmF15CkGy1gh0SxYRyLPlejVc"
            + "fQAI1OU/Pi8uy/tX8\n"
            + "VGqk5vkBZuABKL6eRP8lAOU+9PgIWQQsFcShSNR6WWyroMw"
            + "4+F4LZiQTtNs+UOgM\n"
            + "cZCo1iRZnc4O7JqcGrth/LZdqyE4EXwoXMQRe2BWGbPmdpP"
            + "N+iMAx7YEqrdEZzb8\n"
            + "tDO2n9urXQE6XSR59AFi183bMpbU8Cew+V5zRjoykS/8AhP"
            + "BKA71zTaGZcvyxCtL\n"
            + "ShnQf5sqJ2Np9DhcjDc//iXBCWxzuGAcwEYx9Q1alqEh+oq"
            + "KrJzu1UvpVx0lYykg\n"
            + "5wWc4eYEXou6LFbuoMIt60u/OQeqr7EoPey6Q3SbQ6WFaVB"
            + "pdY4XDGtE6wAp/5GS\n"
            + "k3TZCn8TnHeGGAOVmTos2s3+WHJFivPwZj4Ao5VG/DACo/X"
            + "7EiqXQBKsvQsQCIYn\n"
            + "Mq1EbdiQ9WFG6TIhr6MeFhnNDWZZdptpDdUFbuS72sgrfyx"
            + "9r7C7Wxu/9q/Zg0Vg\n"
            + "VzhEm/W2n8K47xBgIteVjBSKHBKMh90OF2RaHOYFeoOKhEu"
            + "2c6Lkh5ikHW3osQUh\n"
            + "6zzD5WaBLHj7rM2TlVCB3RtUcr/Us2i0MbVaaKKmMnaVLc1"
            + "U6h92Wg0=\n" + "-----END CERTIFICATE-----";

    public static final String DEV_CERT =
        "-----BEGIN CERTIFICATE-----\n"
            + "MIIHCjCCAvKgAwIBAgIBATANBgkqhkiG9w0BAQsFADB/MQs"
            + "wCQYDVQQGEwJCUjEX\n"
            + "MBUGA1UECAwOU2FudGEgQ2F0YXJpbmExFjAUBgNVBAcMDUZ"
            + "sb3JpYW5vcG9saXMx\n"
            + "DjAMBgNVBAoMBVRyYXBwMRQwEgYDVQQLDAtEZXZlbG9wbWV"
            + "udDEZMBcGA1UEAwwQ\n"
            + "RGV2aWNlIFNlcnZlciBDQTAeFw0xNjAzMDEyMjI2NTVaFw0"
            + "xNzAzMDEyMjI2NTVa\n"
            + "MH8xCzAJBgNVBAYTAkJSMRcwFQYDVQQIDA5TYW50YSBDYXR"
            + "hcmluYTEWMBQGA1UE\n"
            + "BwwNRmxvcmlhbm9wb2xpczEOMAwGA1UECgwFVHJhcHAxFDA"
```

```
+ "SBgNVBAsMC0RldmVs\n"
+ "b3BtZW50MRkwFwYDVQQDDBBNb2R1bGUgRGV2ZWxvcGVyMII"
+ "BIjANBgkqhkiG9w0B\n"
+ "AQEFAAOCAQ8AMIIBCgKCAQEAxVT3HkeCJmmb+Wd4njG7iHT"
+ "Y0IX8A/sTq4crE3E0\n"
+ "936TnXsxi07L2nPsNZqx+rF8K5bHM4G8SsQGF4mzIh1U8Kz"
+ "s7qBniDJTXI6s0vsa\n"
+ "EF14sg2kI/RG7RC9wPjMfe4yupbUT6q79YNZ7RZSJU3LNXD"
+ "b68P4E/I4oR5o6Atm\n"
+ "MCSzThHknn+io2PsXntala/39rQzXygibbBWrF1XsH205MP"
+ "xhaR0fUP9cq00nKrT\n"
+ "8Mq+tbaM195I54HPAntMwqOVjH5wQYiAo7HSMJnePARO97Q"
+ "td/uTWoRIlwd84W/D\n"
+ "D2h85F8FEH8KdZAPVLsiqWYyry1xOpjmnwHP0v4J13/2swI"
+ "DAQABo4GQMIGNMAkG\n"
+ "A1UdEwQCMAAwHQYDVR0OBBYEFDTFnKH7gUauBvIOIFniKPj"
+ "smoW1MB8GA1UdIwQY\n"
+ "MBaAFKnucAgamf7uE8Gl9E1zH07SucIVMEAGA1UdHwQ5MDc"
+ "wNaAzoDGGL2h0dHA6\n"
+ "Ly93d3cuaW5mLnVmc2MuYnIvfm1hcmxvbi50cmFwcC9jYS1"
+ "jcmwucGVtMA0GCSqG\n"
+ "SIb3DQEBCwUAA4IEAQCxLwWRgLy0eqRD22FwadTmfOWf8RU"
+ "GHhXCoqDrnbo5MmYm\n"
+ "jwe91LznTmOEFZw53dQP9boCVfmf/WC7kCBd2eIAFs7H561"
+ "d0CK/bHzfgiTlA03D\n"
+ "2S1bTEjbmqzFSnTVSFeXAXawrPwbnb31zwSSi8uwHucGpfw"
+ "5xMOInBPoz6BUznJo\n"
+ "jJiFLwVm0pQrXXIUa9fD9ikSDpqX4v3MFf9vDzatUUb3UzW"
+ "9aVSb2dlt52kUzAPn\n"
+ "PFGSvWDJkMjbcEHvhfdHM0gXCG1liV1qNZdWKpNC0CDpZN/"
+ "/jSgTquD2irDIPYYr\n"
+ "m0fdTDIXiuBEXeyox49gCT+qiTzjaaUtTgmhJPykAJ+rokj"
+ "MeXUPxELCMD8mnxOr\n"
+ "mWyDcnm3jqLY3VLOVE8wNUqZf9x9QuTk0ZqGRZehaQLXDhj"
+ "xMsoe8TaxpTc7jnBm\n"
+ "ScqiWoxWsXLwVZ4AlhPLbVyEqUSE2ZQ7dBaIOCA3pBE0Sue"
+ "XbODXgRlMbP+FfmV3\n"
+ "86Z6k43ZkN4aDV3B2g8gHhD90uq12R0ttuYzF2xdgpssN+l"
+ "6CYVWjcLc7u4t2sJQ\n"
+ "Eo63pUv2tSYoOZshwYJa/cXgcrZv4Dyq+V/Z0nIXy7K9P0f"
+ "qjxpvYcBRjOyHa8H0\n"
+ "rto/C7JrkYJdF199wK82xWmc/RS+URkua9R7q7yjV9qprOx"
+ "Nf+oftDxBPLFo/elL\n"
+ "6cYCmUnFvoO/8LZaOCOFb/A8u+FNJEbIiDWYDFAL74Nv4l1"
+ "BhB9aO9O/f1mJ6EKR\n"
+ "F1soyO+Oc/l3hGBlkj7ELWi3uais+k9B9hPMwUFJKz6waiF"
+ "RhUbcrehLjmnMm2P7\n"
+ "F+Wlu2wwyrFKuF+sMnkeP7igVy/EeK9GlWfHHTMxFclx2NP"
+ "nZKirnT2bx/CPoGJm\n"
+ "YXp4sWEiTf6YyENTqJMXYxoMXT53ZwXqygscOTougtgqFCE"
+ "/t93Z2nhrJDdSaBRk\n"
+ "an8Pz01yGtn9VyoTYM4bJC+nPqrr/de41kkiE37ahKf868l"
```

```
            + "HQG7LsJgtn4CCRYHS\n"
            + "KvPKqHh69cz2igRFU9qZ77GhFBIk08iHnBTwloxHVBpzVT1"
            + "Dz3TpMOiaz9WzkYqO\n"
            + "dB24R0NrcJ5k0gOrptBnziS4v5FzIL2Oh/inL7LH3nGEX5/"
            + "EiG12kgv2fC/s9H40\n"
            + "pErotw6DZjJ1/lkn7qCLbRhHgWEFirZr0wRZBukjYmqad+H"
            + "3R8fO/aodV1vAG8An\n"
            + "LtXmyLJ5svvj/bMFjQCDnscWroLosegXHO3DOaw2PCFKGvm"
            + "+eEQeCek8PfRXls6X\n"
            + "HZ8RcPhl0YkbTBpTLu3EYcgRDXLbXRhBYH+3kYKJBhMKyGG"
            + "fV8VQSv4AEV32kyWM\n"
            + "osGe3D9dc7YXiDYNG2j+OlAgTNOen4M2NCg3HmrX\n"
            + "-----END CERTIFICATE-----";

    public static X509Certificate getCaCert()
        throws CertificateException {
      ByteArrayInputStream byteArrIStream =
          new ByteArrayInputStream(CA_CERT.getBytes());
      CertificateFactory cf =
          CertificateFactory.getInstance("X.509");
      return (X509Certificate) cf
          .generateCertificate(byteArrIStream);
    }

    public static X509Certificate getDevCert()
        throws CertificateException {
      ByteArrayInputStream byteArrIStream =
          new ByteArrayInputStream(DEV_CERT.getBytes());
      CertificateFactory cf =
          CertificateFactory.getInstance("X.509");
      return (X509Certificate) cf
          .generateCertificate(byteArrIStream);
    }

}

package br.com.trapp.deviceserver.model.exception;

public class AuthorizationException extends Exception {

    /**
       *
       */
    private static final long serialVersionUID = 1L;
    public static final AuthorizationException NOT_LOADED =
        new AuthorizationException(
            "Module not installed or loaded");

    public AuthorizationException(String msg,
        Throwable throwable) {
      super(msg, throwable);
    }
```

```
  public AuthorizationException(String msg) {
    super(msg);
  }

}

package br.com.trapp.deviceserver.model.exception;

public class ModuleManagerException extends Exception {

  /**
     *
     */
  private static final long serialVersionUID = 1L;
  public static final ModuleManagerException NOT_A_MODULE =
      new ModuleManagerException(
          "The given jar is not a valid module");

  public ModuleManagerException(String msg,
      Throwable throwable) {
    super(msg, throwable);
  }

  public ModuleManagerException(String msg) {
    super(msg);
  }

}

package br.com.trapp.deviceserver.utils;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URL;
import java.security.cert.CertPath;
import java.security.cert.CertPathValidator;
import java.security.cert.CertStore;
import java.security.cert.CertStoreParameters;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.CollectionCertStoreParameters;
import java.security.cert.PKIXParameters;
import java.security.cert.TrustAnchor;
import java.security.cert.X509CRL;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Collection;
```

```
import java.util.Collections;
import java.util.List;

import org.bouncycastle.asn1.ASN1Primitive;
import org.bouncycastle.asn1.DERIA5String;
import org.bouncycastle.asn1.x509.CRLDistPoint;
import org.bouncycastle.asn1.x509.DistributionPoint;
import org.bouncycastle.asn1.x509.DistributionPointName;
import org.bouncycastle.asn1.x509.Extension;
import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.GeneralNames;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.jce.provider.X509CRLParser;
import org.bouncycastle.x509.extension.X509ExtensionUtil;

import br.com.trapp.deviceserver.model.TrustedCertificates;

public class PKIXUtils {

  /**
   * Verify if the certificate is valid
   *
   * @param pemEncodedCert
   *          The PEM encoded certificate
   *
   * @throws CertificateException
   *           if the certificate is not valid or no encoded
   *           correctly
   */
  public void verifyCertificate(String pemEncodedCert)
      throws CertificateException {
    try {
      ByteArrayInputStream byteArrIStream =
          new ByteArrayInputStream(
              pemEncodedCert.getBytes());
      CertificateFactory cf =
          CertificateFactory.getInstance("X.509");
      this.verifyCertificate((X509Certificate) cf
          .generateCertificate(byteArrIStream));
    } catch (CertificateException e) {
      throw new CertificateException(
          "The certificate is not correctly encoded", e);
    }
  }

  public void verifyCertificate(X509Certificate cert)
      throws CertificateException {
    try {

    } catch (Exception e) {
      throw new CertificateException(
          "The certificate is correctly encoded", e);
```

```
  }
}

/**
 * Verifies the validity of the certificate
 *
 * @param certificate
 * @throws CertificateException
 *           if some error occurred during the
 *           validations, meaning that the certificate is
 *           invalid
 */
@SuppressWarnings("unchecked")
public void verifyCertificateValidity(
    InputStream certificatesInputStream)
    throws CertificateException {
  CertificateFactory cf =
      CertificateFactory.getInstance("X.509");
  Collection<? extends Certificate> certs =
      cf.generateCertificates(certificatesInputStream);
  X509Certificate trust = TrustedCertificates.getCaCert();
  List<URL> urls = this.getCRLURLs(trust);
  if (urls.isEmpty()) {
    throw new CertificateException("None CRL was found");
  }
  Collection<X509CRL> crls = new ArrayList<X509CRL>();
  CertPath cp =
      cf.generateCertPath(new ArrayList<>(certs));
  X509CRLParser parser = new X509CRLParser();
  int index = 0;
  int max = urls.size() - 1;
  try {
    parser.engineInit(urls.get(index).openStream());
    crls.addAll(parser.engineReadAll());
  } catch (Exception e) {
    if (index == max)
      throw new CertificateException(
          "All CRLs are invalids", e);
  }
  this.verifyCertificate(trust, cp, crls);
}

/**
 * Method to get the URL of the CRL from the certificate
 *
 * @param cert
 *          the certificate where the CRLs are
 * @return an list o URL of CRLs
 * @throws CertificateException
 *           if is not possible to get the
 *           cRLDistributionPoints extension
 */
```

```
private List<URL> getCRLURLs(X509Certificate cert)
    throws CertificateException {
  List<URL> crlStreames = new ArrayList<URL>();
  byte[] crlDistPointBytes =
      cert.getExtensionValue(
          Extension.cRLDistributionPoints.getId());
  try {
    ASN1Primitive ext =
        X509ExtensionUtil
            .fromExtensionValue(crlDistPointBytes);
    CRLDistPoint crlsDistPoint =
        CRLDistPoint.getInstance(ext);
    DistributionPoint[] distPoints =
        crlsDistPoint.getDistributionPoints();
    for (DistributionPoint distributionPoint : distPoints) {
      DistributionPointName distPointName =
          distributionPoint.getDistributionPoint();
      GeneralNames generalNames =
          GeneralNames.getInstance(distPointName
              .getName());
      GeneralName[] names = generalNames.getNames();
      for (GeneralName generalName : names) {
        if (generalName.getTagNo() ==
            GeneralName.uniformResourceIdentifier) {
          URI uri =
              new URI(
                  ((DERIA5String) generalName.getName())
                      .getString());
          URL urlFromURI = uri.toURL();
          crlStreames.add(urlFromURI);
        }
      }
    }
  } catch (IOException e) {
    throw new CertificateException(
        "The CRLDistributionPoint extension is invalid",
        e);
  } catch (URISyntaxException e) {
    throw new CertificateException(
        "The URI in CRLDistributionPoint extension is "
        + "invalid",
        e);
  }
  return crlStreames;
}

/**
 * Verify the cert path
 *
 * @param trust
 *          the trusted anchor
 * @param cp
```

```
     *          the cert path to validate
     * @param crls
     *          the list of crls
     *
     * @throws CertificateException
     *           if the certificate path is not valid
     */
    public void verifyCertificate(X509Certificate trust,
        CertPath cp, Collection<X509CRL> crls)
        throws CertificateException {
      try {
        TrustAnchor anchor = new TrustAnchor(trust, null);
        PKIXParameters params =
            new PKIXParameters(Collections.singleton(anchor));
        params.setRevocationEnabled(true);
        CertStoreParameters revoked =
            new CollectionCertStoreParameters(crls);
        params.addCertStore(CertStore.getInstance(
            "Collection", revoked));
        CertPathValidator cpv =
            CertPathValidator.getInstance("PKIX",
                BouncyCastleProvider.PROVIDER_NAME);
        cpv.validate(cp, params);
      } catch (Exception e) {
        throw new CertificateException(
            "The certificate is not valid", e);
      }
    }

}

package br.com.trapp.deviceserver;

import java.io.IOException;

import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerRequestFilter;
import javax.ws.rs.container.ContainerResponseContext;
import javax.ws.rs.container.ContainerResponseFilter;
import javax.ws.rs.container.PreMatching;
import javax.ws.rs.ext.Provider;

@Provider
@PreMatching
public class ConnectionFilter implements
    ContainerRequestFilter, ContainerResponseFilter {

    /**
     * Filter all the requests, do nothing at the time
     */
    public void filter(ContainerRequestContext req)
        throws IOException {
```

```
  }

  /**
   * Filter the responses of the server, add CORS allowing
   */
  public void filter(
      ContainerRequestContext requestContext,
      ContainerResponseContext responseContext)
      throws IOException {

    responseContext.getHeaders().add(
        "Access-Control-Allow-Origin", "*");
    responseContext.getHeaders().add(
        "Access-Control-Allow-Methods",
        "GET, POST, OPTIONS");
    responseContext
        .getHeaders()
        .add(
            "Access-Control-Allow-Headers",
            "Content-Type, Access-Control-Allow-Headers, "
            + "Authorization, X-Requested-With");
  }
}
```

```
package br.com.trapp.deviceserver.controller;

import java.security.cert.CertificateException;
import java.util.Set;

import br.com.trapp.deviceserver.api.Authorization;
import br.com.trapp.deviceserver.api.AuthorizationProccess;
import br.com.trapp.deviceserver.api.DeviceMessage;
import br.com.trapp.deviceserver.model.DeviceBridge;
import br.com.trapp.deviceserver.model.exception.
AuthorizationException;
import br.com.trapp.deviceserver.model.exception.
ModuleManagerException;
import br.com.trapp.deviceserver.model.module.ModuleInfo;
import br.com.trapp.deviceserver.model.module.ModuleManager;

public class MainController {

  private DeviceBridge deviceBridge;

  public MainController() {
    this.deviceBridge = new DeviceBridge();
  }

  public AuthorizationProccess startSession(String module,
      Authorization auth) throws CertificateException,
      AuthorizationException, ModuleManagerException {

    return this.deviceBridge.startSession(module, auth);
```

```
  }

  public Set<ModuleInfo> getAvailableModules() {
    return ModuleManager.getAvailableModules();
  }

  public Integer establishSession(Integer session,
      AuthorizationProccess auth)
      throws AuthorizationException {
    session =
        this.deviceBridge.establishSession(session, auth);
    return session;
  }

  public DeviceMessage sendMessage(Integer session,
      DeviceMessage message) throws AuthorizationException {

    return this.deviceBridge.sendMessage(session, message);
  }

}

package br.com.trapp.deviceserver;

import java.io.File;
import java.io.FilenameFilter;
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.security.Security;

import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.glassfish.grizzly.http.server.HttpServer;
import org.glassfish.grizzly.http.server.NetworkListener;
import org.glassfish.grizzly.nio.transport.TCPNIOTransport;
import org.glassfish.grizzly.threadpool.ThreadPoolConfig;
import org.glassfish.jersey.grizzly2.httpserver.
GrizzlyHttpServerFactory;
import org.glassfish.jersey.server.ResourceConfig;

import br.com.trapp.deviceserver.controller.MainController;
import br.com.trapp.deviceserver.model.exception.
ModuleManagerException;
import br.com.trapp.deviceserver.model.module.ModuleManager;

//public class Main extends Application {
public class Main {
  private static final String MODULES_FOLDER = "/modules/";
  // Base URI the Grizzly HTTP server will listen on
  public static final String BASE_URI =
```

```
    "http://localhost:9977/deviceserver/";
// private static MainWindow mainWindow;
private static MainController mainContoller;

/**
 * Main method.
 *
 * @param args
 * @throws IOException
 * @throws ModuleManagerException
 * @throws URISyntaxException
 */
public static void main(String[] args)
    throws IOException, ModuleManagerException,
    URISyntaxException {

  Path path =
      Paths.get(Main.class.getProtectionDomain()
          .getCodeSource().getLocation().toURI());
  String dsJarFile = path.toString();
  String modulesFolderPath =
      dsJarFile.substring(0,
          dsJarFile.lastIndexOf(File.separator))
          + MODULES_FOLDER;
  File modulesDir = new File(modulesFolderPath);
  if (modulesDir.exists()) {
    Security.addProvider(new BouncyCastleProvider());
    // Create server
    final HttpServer server = startServer();
    FilenameFilter filenameFilter = new FilenameFilter() {
      @Override
      public boolean accept(File dir, String name) {
        return name.endsWith(".jar");
      }
    };
    String[] modules = modulesDir.list(filenameFilter);
    for (String moduleFilename : modules) {
      System.out.println("Loading module: "
          + moduleFilename);
      File module =
          new File(modulesFolderPath + File.separator
              + moduleFilename);
      ModuleManager.loadModule(module);
      System.out.println("Module loaded: "
          + moduleFilename);
    }

    System.out.println(String.format(
        "Jersey app started with WADL available at "
            + "%sapplication.wadl\nHit enter to "
            + "stop it...", BASE_URI));
```

```
        mainContoller = new MainController();
        ServerResource.controller = mainContoller;
        System.in.read();
        server.shutdownNow();
      } else {
        System.out
            .println("Exiting, modules folder not found");
      }
      System.exit(0);
    }

    /**
     * Starts Grizzly HTTP server exposing JAX-RS resources
     * defined in this application.
     *
     * @return Grizzly HTTP server.
     */
    public static HttpServer startServer() {

      try {
        final ResourceConfig rc =
            new ResourceConfig(ServerResource.class);
        rc.register(ConnectionFilter.class);

        HttpServer server =
            GrizzlyHttpServerFactory.createHttpServer(
                URI.create(BASE_URI), rc, false);

        // create the thread pool configuration
        // reconfigure the thread pool
        // Setting all thread pool with 1, because the
        // purpose of this
        // application
        // is to have few connections only
        NetworkListener listener =
            server.getListeners().iterator().next();
        TCPNIOTransport transport = listener.getTransport();
        ThreadPoolConfig workerThreadPoll =
            transport.getWorkerThreadPoolConfig();
        workerThreadPoll.setCorePoolSize(1);
        workerThreadPoll.setMaxPoolSize(2);
        transport.setSelectorRunnersCount(1);
        server.start();
        return server;
      } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
      }
      return null;
    }
}
```

```
package br.com.trapp.deviceserver;

import java.security.cert.CertificateException;
import java.util.Set;

import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

import br.com.trapp.deviceserver.api.Authorization;
import br.com.trapp.deviceserver.api.AuthorizationProccess;
import br.com.trapp.deviceserver.api.DeviceMessage;
import br.com.trapp.deviceserver.controller.MainController;
import br.com.trapp.deviceserver.model.exception.
AuthorizationException;
import br.com.trapp.deviceserver.model.exception.
ModuleManagerException;
import br.com.trapp.deviceserver.model.module.ModuleInfo;

@Path("")
public class ServerResource {

  public static MainController controller;

  @GET
  @Path("/")
  @Produces(MediaType.APPLICATION_JSON)
  public Set<ModuleInfo> getAvailableDevices() {
    return controller.getAvailableModules();
  }

  @POST
  @Path("startsession/{module}")
  @Consumes(MediaType.APPLICATION_JSON)
  @Produces(MediaType.APPLICATION_JSON)
  public AuthorizationProccess
      startSession(@PathParam("module") String module,
          Authorization auth) {
    try {
      return controller.startSession(module, auth);
    } catch (CertificateException e) {
      e.printStackTrace();
      throw new WebApplicationException(
          Response
              .status(500)
              .entity(
```

```
                "Please check if the given certificate is"
                + " right encoded in PEM format")
            .build());
  } catch (AuthorizationException e) {
    throw new WebApplicationException(Response
        .status(500)
        .entity("The requested module is not loaded")
        .build());
  } catch (ModuleManagerException e) {
    throw new WebApplicationException(
        Response
            .status(500)
            .entity(
                "The requested module was not recognized")
            .build());
  } catch (Throwable e) {
    e.printStackTrace();
    throw e;
  }
}

@POST
@Path("establishsession/{session}")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Integer establishSession(
    @PathParam("session") Integer session,
    AuthorizationProccess auth) {
  try {
    return controller.establishSession(session, auth);
  } catch (AuthorizationException e) {
    e.printStackTrace();
    throw new WebApplicationException(Response
        .status(500).entity(e.getMessage()).build());
  }
}

@POST
@Path("{session}")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public DeviceMessage sendMessage(
    @PathParam("session") Integer session,
    DeviceMessage message) {
  try {
    return controller.sendMessage(session, message);
  } catch (AuthorizationException e) {
    e.printStackTrace();
    throw new WebApplicationException(Response
        .status(500).entity(e.getMessage()).build());
  }
}
```

```
}
```

ANEXO C – Código fonte Echo module

```
package br.com.trapp.deviceserver.echomodule;

import br.com.trapp.deviceserver.api.DeviceMessage;
import br.com.trapp.deviceserver.api.DeviceModule;

public class EchoModule implements DeviceModule {

  public DeviceMessage
      incomingMessage(DeviceMessage message) {
    switch (message.getMethod()) {
    case "echoreverse":
      StringBuilder sb =
          new StringBuilder(message.getMessage());
      sb.reverse();
      message.setMessage(sb.toString());
      break;
    }
    return message;
  }

  public String getName() {
    return "Echo Module";
  }

  public String getVersion() {
    return "1.0";
  }

  public String getIdendifier() {
    return "echo";
  }

}
```

ANEXO D – Código fonte Cryptographic module

```
package br.com.trapp.deviceserver.cryptographicmodule.
module;

import java.io.IOException;
import java.util.Base64;
import java.util.List;
import java.util.Set;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

import br.com.trapp.deviceserver.api.DeviceException;
import br.com.trapp.deviceserver.api.DeviceMessage;
import br.com.trapp.deviceserver.api.DeviceModule;
import br.com.trapp.deviceserver.cryptographicmodule.
objects.CertificateInfo;
import br.com.trapp.deviceserver.cryptographicmodule.
objects.CryptoDeviceInfo;
import br.com.trapp.deviceserver.cryptographicmodule.
objects.CryptoDevices;
import br.com.trapp.deviceserver.cryptographicmodule.
objects.SignatureRequest;

public class CryptographicModule implements DeviceModule {

  public DeviceMessage
      incomingMessage(DeviceMessage message)
          throws DeviceException {
    switch (message.getMethod()) {
    case "list":
      return this.list();
    case "listCerts":
      return this.listCerts(message);
    case "sign":
      return this.sign(message);
    case "verify":
      return this.verify(message);
    default:
      break;
    }
    return new DeviceMessage("return", "result: {}");
  }

  public String getName() {
    return "Cryptographic Module";
  }

  public String getVersion() {
    return "1.0";
```

```
}

public String getIdendifier() {
  return "crypto";
}

/**
 * List all connected devices
 *
 * @return a list with label and serial of all connected
 *         devices
 */
private DeviceMessage list() {
  Set<CryptoDeviceInfo> devices = CryptoDevices.list();
  ObjectMapper mapper = new ObjectMapper();
  // Object to JSON in String
  String jsonInString = null;
  try {
    jsonInString = mapper.writeValueAsString(devices);
  } catch (JsonProcessingException e) {
    e.printStackTrace();
  }
  return new DeviceMessage("list", jsonInString);
}

/**
 * List all certs inside the given device
 *
 * @param message
 *          the message
 * @return the certs with public keys in this device
 */
private DeviceMessage listCerts(DeviceMessage message) {
  String jsonInString = null;
  try {
    ObjectMapper mapper = new ObjectMapper();
    CryptoDeviceInfo deviceInfo = null;
    deviceInfo =
        mapper.readValue(message.getMessage(),
            CryptoDeviceInfo.class);
    List<CertificateInfo> certs =
        CryptoDevices
            .listCertsWithMatchingKey(deviceInfo);

    // Object to JSON in String
    jsonInString = mapper.writeValueAsString(certs);
  } catch (IOException e) {
    e.printStackTrace();
  }
  return new DeviceMessage("listCerts", jsonInString);
}
```

```
/**
 * Sign a message
 *
 * @param message
 * @return
 * @throws DeviceException
 */
private DeviceMessage sign(DeviceMessage message)
    throws DeviceException {
  String jsonInString = null;
  try {
    ObjectMapper mapper = new ObjectMapper();
    SignatureRequest signReq = null;
    signReq =
        mapper.readValue(message.getMessage(),
            SignatureRequest.class);
    byte[] signedBytes = CryptoDevices.sign(signReq);

    // Object to JSON in String
    jsonInString =
        mapper.writeValueAsString(Base64.getEncoder()
            .encodeToString(signedBytes));
  } catch (IOException e) {
    throw new DeviceException(e.getMessage(), e);
  }
  return new DeviceMessage("sign", jsonInString);
}

/**
 * Verify a message
 *
 * @param message
 * @return
 * @throws Exception
 */
private DeviceMessage verify(DeviceMessage message)
    throws DeviceException {
  String jsonInString = null;
  try {
    ObjectMapper mapper = new ObjectMapper();
    SignatureRequest signReq = null;
    signReq =
        mapper.readValue(message.getMessage(),
            SignatureRequest.class);
    byte[] signedBytes = CryptoDevices.sign(signReq);

    // Object to JSON in String
    jsonInString =
        mapper.writeValueAsString(Base64.getEncoder()
            .encodeToString(signedBytes));
  } catch (IOException e) {
    throw new DeviceException(e.getMessage(), e);
```

```
    }
    return new DeviceMessage("verify", jsonInString);
  }

}

package br.com.trapp.deviceserver.cryptographicmodule.
objects;

public class CertificateInfo {

  private String id;
  private String label;
  private String deviceSerial;

  public CertificateInfo() {
  }

  public CertificateInfo(String id, String label,
      String deviceSerial) {
    this.id = id;
    this.label = label;
    this.deviceSerial = deviceSerial;
  }

  public String getId() {
    return id;
  }

  public void setId(String id) {
    this.id = id;
  }

  public String getLabel() {
    return label;
  }

  public void setLabel(String label) {
    this.label = label;
  }

  public String getDeviceSerial() {
    return deviceSerial;
  }

  public void setDeviceSerial(String deviceSerial) {
    this.deviceSerial = deviceSerial;
  }

  @Override
  public int hashCode() {
    final int prime = 31;
```

```
    int result = 1;
    result =
        prime * result + ((id == null) ? 0 : id.hashCode());
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    CertificateInfo other = (CertificateInfo) obj;
    if (id == null) {
      if (other.id != null)
        return false;
    } else if (!id.equals(other.id))
      return false;
    return true;
  }

}

package br.com.trapp.deviceserver.cryptographicmodule.
objects;

import javax.xml.bind.DatatypeConverter;

import iaik.pkcs.pkcs11.objects.Attribute;
import iaik.pkcs.pkcs11.objects.ByteArrayAttribute;
import iaik.pkcs.pkcs11.objects.CharArrayAttribute;
import iaik.pkcs.pkcs11.objects.X509PublicKeyCertificate;

public class CertificateTemplate extends
    X509PublicKeyCertificate {

  public CertificateTemplate() {
    super();
  }

  @SuppressWarnings("unchecked")
  public void setIdAttribute(String value) {
    ByteArrayAttribute attr =
        new ByteArrayAttribute(Attribute.ID);
    attr.setByteArrayValue(DatatypeConverter
        .parseHexBinary(value));
    attributeTable_.put(Attribute.ID, attr);
  }

  @SuppressWarnings("unchecked")
```

```
  public void setLabelAttribute(String value) {
    CharArrayAttribute attr =
        new CharArrayAttribute(Attribute.LABEL);
    attr.setCharArrayValue(value.toCharArray());
    attributeTable_.put(Attribute.LABEL, attr);
  }
}

package br.com.trapp.deviceserver.cryptographicmodule.
objects;

public class CryptoDeviceInfo {

  private String serial;
  private String label;

  public CryptoDeviceInfo() {
  }

  public CryptoDeviceInfo(String serial, String label) {
    this.serial = serial;
    this.label = label;
  }

  public String getSerial() {
    return serial;
  }

  public void setSerial(String serial) {
    this.serial = serial;
  }

  public String getLabel() {
    return label;
  }

  public void setLabel(String label) {
    this.label = label;
  }

  @Override
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result =
        prime * result
            + ((serial == null) ? 0 : serial.hashCode());
    return result;
  }

  @Override
  public boolean equals(Object obj) {
```

```
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    CryptoDeviceInfo other = (CryptoDeviceInfo) obj;
    if (serial == null) {
      if (other.serial != null)
        return false;
    } else if (!serial.equals(other.serial))
      return false;
    return true;
  }

  @Override
  public String toString() {
    return "CryptoDeviceInfo [serial=" + serial
        + ", label=" + label + "]";
  }

}

package br.com.trapp.deviceserver.cryptographicmodule.
objects;

public class SignatureRequest {

  private String tokenSerial;
  private String keyLabel;
  private String keyId;
  private String hashAlgorithm;
  private String data;

  public SignatureRequest() {
  }

  public SignatureRequest(String tokenSerial,
      String keyLabel, String keyId, String hashAlgorithm,
      String data) {
    this.tokenSerial = tokenSerial;
    this.keyLabel = keyLabel;
    this.keyId = keyId;
    this.hashAlgorithm = hashAlgorithm;
    this.data = data;
  }

  public String getKeyLabel() {
    return keyLabel;
  }

  public void setKeyLabel(String keyLabel) {
```

```
    this.keyLabel = keyLabel;
  }

  public String getKeyId() {
    return keyId;
  }

  public void setKeyId(String keyId) {
    this.keyId = keyId;
  }

  public String getHashAlgorithm() {
    return hashAlgorithm;
  }

  public void setHashAlgorithm(String hashAlgorithm) {
    this.hashAlgorithm = hashAlgorithm;
  }

  public String getTokenSerial() {
    return tokenSerial;
  }

  public void setTokenSerial(String tokenSerial) {
    this.tokenSerial = tokenSerial;
  }

  public String getData() {
    return data;
  }

  public void setData(String data) {
    this.data = data;
  }

  @Override
  public String toString() {
    return "SignatureRequest [tokenSerial=" + tokenSerial
        + ", keyLabel=" + keyLabel + ", keyId=" + keyId
        + ", hashAlgorithm=" + hashAlgorithm + ", data="
        + data + "]";
  }

}

package br.com.trapp.deviceserver.cryptographicmodule.
objects;

import java.util.HashMap;
import java.util.Map;

import iaik.pkcs.pkcs11.Mechanism;
```

```
import iaik.pkcs.pkcs11.objects.DSAPrivateKey;
import iaik.pkcs.pkcs11.objects.ECDSAPrivateKey;
import iaik.pkcs.pkcs11.objects.KEAPrivateKey;
import iaik.pkcs.pkcs11.objects.PrivateKey;
import iaik.pkcs.pkcs11.objects.RSAPrivateKey;

public class Mechanisms {

  static final long CKM_RSA_PKCS_KEY_PAIR_GEN = 0x00000000;
  static final long CKM_RSA_PKCS = 0x00000001;
  static final long CKM_RSA_9796 = 0x00000002;
  static final long CKM_RSA_X_509 = 0x00000003;
  static final long CKM_MD2_RSA_PKCS = 0x00000004;
  static final long CKM_MD5_RSA_PKCS = 0x00000005;
  static final long CKM_SHA1_RSA_PKCS = 0x00000006;
  static final long CKM_RIPEMD128_RSA_PKCS = 0x00000007;
  static final long CKM_RIPEMD160_RSA_PKCS = 0x00000008;
  static final long CKM_RSA_PKCS_OAEP = 0x00000009;
  static final long CKM_RSA_X9_31_KEY_PAIR_GEN = 0x0000000A;
  static final long CKM_RSA_X9_31 = 0x0000000B;
  static final long CKM_SHA1_RSA_X9_31 = 0x0000000C;
  static final long CKM_RSA_PKCS_PSS = 0x0000000D;
  static final long CKM_SHA1_RSA_PKCS_PSS = 0x0000000E;
  static final long CKM_DSA_KEY_PAIR_GEN = 0x00000010;
  static final long CKM_DSA = 0x00000011;
  static final long CKM_DSA_SHA1 = 0x00000012;
  static final long CKM_DSA_FIPS_G_GEN = 0x00000013;
  static final long CKM_DSA_SHA224 = 0x00000014;
  static final long CKM_DSA_SHA256 = 0x00000015;
  static final long CKM_DSA_SHA384 = 0x00000016;
  static final long CKM_DSA_SHA512 = 0x00000017;
  static final long CKM_DH_PKCS_KEY_PAIR_GEN = 0x00000020;
  static final long CKM_DH_PKCS_DERIVE = 0x00000021;
  static final long CKM_X9_42_DH_KEY_PAIR_GEN = 0x00000030;
  static final long CKM_X9_42_DH_DERIVE = 0x00000031;
  static final long CKM_X9_42_DH_HYBRID_DERIVE = 0x00000032;
  static final long CKM_X9_42_MQV_DERIVE = 0x00000033;
  static final long CKM_SHA256_RSA_PKCS = 0x00000040;
  static final long CKM_SHA384_RSA_PKCS = 0x00000041;
  static final long CKM_SHA512_RSA_PKCS = 0x00000042;
  static final long CKM_SHA256_RSA_PKCS_PSS = 0x00000043;
  static final long CKM_SHA384_RSA_PKCS_PSS = 0x00000044;
  static final long CKM_SHA512_RSA_PKCS_PSS = 0x00000045;
  static final long CKM_SHA224_RSA_PKCS = 0x00000046;
  static final long CKM_SHA224_RSA_PKCS_PSS = 0x00000047;
  static final long CKM_SHA512_224 = 0x00000048;
  static final long CKM_SHA512_224_HMAC = 0x00000049;
  static final long CKM_SHA512_224_HMAC_GENERAL =
      0x0000004A;
  static final long CKM_SHA512_224_KEY_DERIVATION =
      0x0000004B;
  static final long CKM_SHA512_256 = 0x0000004C;
```

```
    static final long CKM_SHA512_256_HMAC = 0x0000004D;
    static final long CKM_SHA512_256_HMAC_GENERAL =
        0x0000004E;
    static final long CKM_SHA512_256_KEY_DERIVATION =
        0x0000004F;
    static final long CKM_SHA512_T = 0x00000050;
    static final long CKM_SHA512_T_HMAC = 0x00000051;
    static final long CKM_SHA512_T_HMAC_GENERAL = 0x00000052;
    static final long CKM_SHA512_T_KEY_DERIVATION =
        0x00000053;
    static final long CKM_RC2_KEY_GEN = 0x00000100;
    static final long CKM_RC2_ECB = 0x00000101;
    static final long CKM_RC2_CBC = 0x00000102;
    static final long CKM_RC2_MAC = 0x00000103;
    static final long CKM_RC2_MAC_GENERAL = 0x00000104;
    static final long CKM_RC2_CBC_PAD = 0x00000105;
    static final long CKM_RC4_KEY_GEN = 0x00000110;
    static final long CKM_RC4 = 0x00000111;
    static final long CKM_DES_KEY_GEN = 0x00000120;
    static final long CKM_DES_ECB = 0x00000121;
    static final long CKM_DES_CBC = 0x00000122;
    static final long CKM_DES_MAC = 0x00000123;
    static final long CKM_DES_MAC_GENERAL = 0x00000124;
    static final long CKM_DES_CBC_PAD = 0x00000125;
    static final long CKM_DES2_KEY_GEN = 0x00000130;
    static final long CKM_DES3_KEY_GEN = 0x00000131;
    static final long CKM_DES3_ECB = 0x00000132;
    static final long CKM_DES3_CBC = 0x00000133;
    static final long CKM_DES3_MAC = 0x00000134;
    static final long CKM_DES3_MAC_GENERAL = 0x00000135;
    static final long CKM_DES3_CBC_PAD = 0x00000136;
    static final long CKM_CDMF_KEY_GEN = 0x00000140;
    static final long CKM_CDMF_ECB = 0x00000141;
    static final long CKM_CDMF_CBC = 0x00000142;
    static final long CKM_CDMF_MAC = 0x00000143;
    static final long CKM_CDMF_MAC_GENERAL = 0x00000144;
    static final long CKM_CDMF_CBC_PAD = 0x00000145;
    static final long CKM_DES_OFB64 = 0x00000150;
    static final long CKM_DES_OFB8 = 0x00000151;
    static final long CKM_DES_CFB64 = 0x00000152;
    static final long CKM_DES_CFB8 = 0x00000153;
    static final long CKM_MD2 = 0x00000200;
    static final long CKM_MD2_HMAC = 0x00000201;
    static final long CKM_MD2_HMAC_GENERAL = 0x00000202;
    static final long CKM_MD5 = 0x00000210;
    static final long CKM_MD5_HMAC = 0x00000211;
    static final long CKM_MD5_HMAC_GENERAL = 0x00000212;
    static final long CKM_SHA_1 = 0x00000220;
    static final long CKM_SHA_1_HMAC = 0x00000221;
    static final long CKM_SHA_1_HMAC_GENERAL = 0x00000222;
    static final long CKM_RIPEMD128 = 0x00000230;
    static final long CKM_RIPEMD128_HMAC = 0x00000231;
```

```
static final long CKM_RIPEMD128_HMAC_GENERAL = 0x00000232;
static final long CKM_RIPEMD160 = 0x00000240;
static final long CKM_RIPEMD160_HMAC = 0x00000241;
static final long CKM_RIPEMD160_HMAC_GENERAL = 0x00000242;
static final long CKM_SHA256 = 0x00000250;
static final long CKM_SHA256_HMAC = 0x00000251;
static final long CKM_SHA256_HMAC_GENERAL = 0x00000252;
static final long CKM_SHA224 = 0x00000255;
static final long CKM_SHA224_HMAC = 0x00000256;
static final long CKM_SHA224_HMAC_GENERAL = 0x00000257;
static final long CKM_SHA384 = 0x00000260;
static final long CKM_SHA384_HMAC = 0x00000261;
static final long CKM_SHA384_HMAC_GENERAL = 0x00000262;
static final long CKM_SHA512 = 0x00000270;
static final long CKM_SHA512_HMAC = 0x00000271;
static final long CKM_SHA512_HMAC_GENERAL = 0x00000272;
static final long CKM_SECURID_KEY_GEN = 0x00000280;
static final long CKM_SECURID = 0x00000282;
static final long CKM_HOTP_KEY_GEN = 0x00000290;
static final long CKM_HOTP = 0x00000291;
static final long CKM_ACTI = 0x000002A0;
static final long CKM_ACTI_KEY_GEN = 0x000002A1;
static final long CKM_CAST_KEY_GEN = 0x00000300;
static final long CKM_CAST_ECB = 0x00000301;
static final long CKM_CAST_CBC = 0x00000302;
static final long CKM_CAST_MAC = 0x00000303;
static final long CKM_CAST_MAC_GENERAL = 0x00000304;
static final long CKM_CAST_CBC_PAD = 0x00000305;
static final long CKM_CAST3_KEY_GEN = 0x00000310;
static final long CKM_CAST3_ECB = 0x00000311;
static final long CKM_CAST3_CBC = 0x00000312;
static final long CKM_CAST3_MAC = 0x00000313;
static final long CKM_CAST3_MAC_GENERAL = 0x00000314;
static final long CKM_CAST3_CBC_PAD = 0x00000315;
static final long CKM_CAST5_KEY_GEN = 0x00000320;
static final long CKM_CAST128_KEY_GEN = 0x00000320;
static final long CKM_CAST5_ECB = 0x00000321;
static final long CKM_CAST128_ECB = 0x00000321;
static final long CKM_CAST5_CBC = 0x00000322;
static final long CKM_CAST128_CBC = 0x00000322;
static final long CKM_CAST5_MAC = 0x00000323;
static final long CKM_CAST128_MAC = 0x00000323;
static final long CKM_CAST5_MAC_GENERAL = 0x00000324;
static final long CKM_CAST128_MAC_GENERAL = 0x00000324;
static final long CKM_CAST5_CBC_PAD = 0x00000325;
static final long CKM_CAST128_CBC_PAD = 0x00000325;
static final long CKM_RC5_KEY_GEN = 0x00000330;
static final long CKM_RC5_ECB = 0x00000331;
static final long CKM_RC5_CBC = 0x00000332;
static final long CKM_RC5_MAC = 0x00000333;
static final long CKM_RC5_MAC_GENERAL = 0x00000334;
static final long CKM_RC5_CBC_PAD = 0x00000335;
```

```
static final long CKM_IDEA_KEY_GEN = 0x00000340;
static final long CKM_IDEA_ECB = 0x00000341;
static final long CKM_IDEA_CBC = 0x00000342;
static final long CKM_IDEA_MAC = 0x00000343;
static final long CKM_IDEA_MAC_GENERAL = 0x00000344;
static final long CKM_IDEA_CBC_PAD = 0x00000345;
static final long CKM_GENERIC_SECRET_KEY_GEN = 0x00000350;
static final long CKM_CONCATENATE_BASE_AND_KEY =
    0x00000360;
static final long CKM_CONCATENATE_BASE_AND_DATA =
    0x00000362;
static final long CKM_CONCATENATE_DATA_AND_BASE =
    0x00000363;
static final long CKM_XOR_BASE_AND_DATA = 0x00000364;
static final long CKM_EXTRACT_KEY_FROM_KEY = 0x00000365;
static final long CKM_SSL3_PRE_MASTER_KEY_GEN =
    0x00000370;
static final long CKM_SSL3_MASTER_KEY_DERIVE = 0x00000371;
static final long CKM_SSL3_KEY_AND_MAC_DERIVE =
    0x00000372;
static final long CKM_SSL3_MASTER_KEY_DERIVE_DH =
    0x00000373;
static final long CKM_TLS_PRE_MASTER_KEY_GEN = 0x00000374;
static final long CKM_TLS_MASTER_KEY_DERIVE = 0x00000375;
static final long CKM_TLS_KEY_AND_MAC_DERIVE = 0x00000376;
static final long CKM_TLS_MASTER_KEY_DERIVE_DH =
    0x00000377;
static final long CKM_TLS_PRF = 0x00000378;
static final long CKM_SSL3_MD5_MAC = 0x00000380;
static final long CKM_SSL3_SHA1_MAC = 0x00000381;
static final long CKM_MD5_KEY_DERIVATION = 0x00000390;
static final long CKM_MD2_KEY_DERIVATION = 0x00000391;
static final long CKM_SHA1_KEY_DERIVATION = 0x00000392;
static final long CKM_SHA256_KEY_DERIVATION = 0x00000393;
static final long CKM_SHA384_KEY_DERIVATION = 0x00000394;
static final long CKM_SHA512_KEY_DERIVATION = 0x00000395;
static final long CKM_SHA224_KEY_DERIVATION = 0x00000396;
static final long CKM_PBE_MD2_DES_CBC = 0x000003A0;
static final long CKM_PBE_MD5_DES_CBC = 0x000003A1;
static final long CKM_PBE_MD5_CAST_CBC = 0x000003A2;
static final long CKM_PBE_MD5_CAST3_CBC = 0x000003A3;
static final long CKM_PBE_MD5_CAST5_CBC = 0x000003A4;
static final long CKM_PBE_MD5_CAST128_CBC = 0x000003A4;
static final long CKM_PBE_SHA1_CAST5_CBC = 0x000003A5;
static final long CKM_PBE_SHA1_CAST128_CBC = 0x000003A5;
static final long CKM_PBE_SHA1_RC4_128 = 0x000003A6;
static final long CKM_PBE_SHA1_RC4_40 = 0x000003A7;
static final long CKM_PBE_SHA1_DES3_EDE_CBC = 0x000003A8;
static final long CKM_PBE_SHA1_DES2_EDE_CBC = 0x000003A9;
static final long CKM_PBE_SHA1_RC2_128_CBC = 0x000003AA;
static final long CKM_PBE_SHA1_RC2_40_CBC = 0x000003AB;
static final long CKM_PKCS5_PBKD2 = 0x000003B0;
```

```
static final long CKM_PBA_SHA1_WITH_SHA1_HMAC =
    0x000003C0;
static final long CKM_WTLS_PRE_MASTER_KEY_GEN =
    0x000003D0;
static final long CKM_WTLS_MASTER_KEY_DERIVE = 0x000003D1;
static final long CKM_WTLS_MASTER_KEY_DERIVE_DH_ECC =
    0x000003D2;
static final long CKM_WTLS_PRF = 0x000003D3;
static final long CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE =
    0x000003D4;
static final long CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE =
    0x000003D5;
static final long CKM_TLS10_MAC_SERVER = 0x000003D6;
static final long CKM_TLS10_MAC_CLIENT = 0x000003D7;
static final long CKM_TLS12_MAC = 0x000003D8;
static final long CKM_TLS12_MASTER_KEY_DERIVE =
    0x000003E0;
static final long CKM_TLS12_KEY_AND_MAC_DERIVE =
    0x000003E1;
static final long CKM_TLS12_MASTER_KEY_DERIVE_DH =
    0x000003E2;
static final long CKM_TLS12_KEY_SAFE_DERIVE = 0x000003E3;
static final long CKM_TLS_MAC = 0x000003E4;
static final long CKM_TLS_KDF = 0x000003E5;
static final long CKM_KEY_WRAP_LYNKS = 0x00000400;
static final long CKM_KEY_WRAP_SET_OAEP = 0x00000401;
static final long CKM_CMS_SIG = 0x00000500;
static final long CKM_KIP_DERIVE = 0x00000510;
static final long CKM_KIP_WRAP = 0x00000511;
static final long CKM_KIP_MAC = 0x00000512;
static final long CKM_CAMELLIA_KEY_GEN = 0x00000550;
static final long CKM_CAMELLIA_ECB = 0x00000551;
static final long CKM_CAMELLIA_CBC = 0x00000552;
static final long CKM_CAMELLIA_MAC = 0x00000553;
static final long CKM_CAMELLIA_MAC_GENERAL = 0x00000554;
static final long CKM_CAMELLIA_CBC_PAD = 0x00000555;
static final long CKM_CAMELLIA_ECB_ENCRYPT_DATA =
    0x00000556;
static final long CKM_CAMELLIA_CBC_ENCRYPT_DATA =
    0x00000557;
static final long CKM_CAMELLIA_CTR = 0x00000558;
static final long CKM_ARIA_KEY_GEN = 0x00000560;
static final long CKM_ARIA_ECB = 0x00000561;
static final long CKM_ARIA_CBC = 0x00000562;
static final long CKM_ARIA_MAC = 0x00000563;
static final long CKM_ARIA_MAC_GENERAL = 0x00000564;
static final long CKM_ARIA_CBC_PAD = 0x00000565;
static final long CKM_ARIA_ECB_ENCRYPT_DATA = 0x00000566;
static final long CKM_ARIA_CBC_ENCRYPT_DATA = 0x00000567;
static final long CKM_SKIPJACK_KEY_GEN = 0x00001000;
static final long CKM_SKIPJACK_ECB64 = 0x00001001;
static final long CKM_SKIPJACK_CBC64 = 0x00001002;
```

```
static final long CKM_SKIPJACK_OFB64 = 0x00001003;
static final long CKM_SKIPJACK_CFB64 = 0x00001004;
static final long CKM_SKIPJACK_CFB32 = 0x00001005;
static final long CKM_SKIPJACK_CFB16 = 0x00001006;
static final long CKM_SKIPJACK_CFB8 = 0x00001007;
static final long CKM_SKIPJACK_WRAP = 0x00001008;
static final long CKM_SKIPJACK_PRIVATE_WRAP = 0x00001009;
static final long CKM_SKIPJACK_RELAYX = 0x0000100a;
static final long CKM_KEA_KEY_PAIR_GEN = 0x00001010;
static final long CKM_KEA_KEY_DERIVE = 0x00001011;
static final long CKM_FORTEZZA_TIMESTAMP = 0x00001020;
static final long CKM_BATON_KEY_GEN = 0x00001030;
static final long CKM_BATON_ECB128 = 0x00001031;
static final long CKM_BATON_ECB96 = 0x00001032;
static final long CKM_BATON_CBC128 = 0x00001033;
static final long CKM_BATON_COUNTER = 0x00001034;
static final long CKM_BATON_SHUFFLE = 0x00001035;
static final long CKM_BATON_WRAP = 0x00001036;
static final long CKM_ECDSA_KEY_PAIR_GEN = 0x00001040;
static final long CKM_EC_KEY_PAIR_GEN = 0x00001040;
static final long CKM_ECDSA = 0x00001041;
static final long CKM_ECDSA_SHA1 = 0x00001042;
static final long CKM_ECDH1_DERIVE = 0x00001050;
static final long CKM_ECDH1_COFACTOR_DERIVE = 0x00001051;
static final long CKM_ECMQV_DERIVE = 0x00001052;
static final long CKM_ECDH_AES_KEY_WRAP = 0x00001053;
static final long CKM_RSA_AES_KEY_WRAP = 0x00001054;
static final long CKM_JUNIPER_KEY_GEN = 0x00001060;
static final long CKM_JUNIPER_ECB128 = 0x00001061;
static final long CKM_JUNIPER_CBC128 = 0x00001062;
static final long CKM_JUNIPER_COUNTER = 0x00001063;
static final long CKM_JUNIPER_SHUFFLE = 0x00001064;
static final long CKM_JUNIPER_WRAP = 0x00001065;
static final long CKM_FASTHASH = 0x00001070;
static final long CKM_AES_KEY_GEN = 0x00001080;
static final long CKM_AES_ECB = 0x00001081;
static final long CKM_AES_CBC = 0x00001082;
static final long CKM_AES_MAC = 0x00001083;
static final long CKM_AES_MAC_GENERAL = 0x00001084;
static final long CKM_AES_CBC_PAD = 0x00001085;
static final long CKM_AES_CTR = 0x00001086;
static final long CKM_AES_GCM = 0x00001087;
static final long CKM_AES_CCM = 0x00001088;
static final long CKM_AES_CMAC_GENERAL = 0x00001089;
static final long CKM_AES_CMAC = 0x0000108A;
static final long CKM_AES_CTS = 0x0000108B;
static final long CKM_AES_XCBC_MAC = 0x0000108C;
static final long CKM_AES_XCBC_MAC_96 = 0x0000108D;
static final long CKM_BLOWFISH_KEY_GEN = 0x00001090;
static final long CKM_BLOWFISH_CBC = 0x00001091;
static final long CKM_TWOFISH_KEY_GEN = 0x00001092;
static final long CKM_TWOFISH_CBC = 0x00001093;
```

```
static final long CKM_BLOWFISH_CBC_PAD = 0x00001094;
static final long CKM_TWOFISH_CBC_PAD = 0x00001095;
static final long CKM_DES_ECB_ENCRYPT_DATA = 0x00001100;
static final long CKM_DES_CBC_ENCRYPT_DATA = 0x00001101;
static final long CKM_DES3_ECB_ENCRYPT_DATA = 0x00001102;
static final long CKM_DES3_CBC_ENCRYPT_DATA = 0x00001103;
static final long CKM_AES_ECB_ENCRYPT_DATA = 0x00001104;
static final long CKM_AES_CBC_ENCRYPT_DATA = 0x00001105;
static final long CKM_GOSTR3410_KEY_PAIR_GEN = 0x00001200;
static final long CKM_GOSTR3410 = 0x00001201;
static final long CKM_GOSTR3410_WITH_GOSTR3411 =
    0x00001202;
static final long CKM_GOSTR3410_KEY_WRAP = 0x00001203;
static final long CKM_GOSTR3410_DERIVE = 0x00001204;
static final long CKM_GOSTR3411 = 0x00001210;
static final long CKM_GOSTR3411_HMAC = 0x00001211;
static final long CKM_GOST28147_KEY_GEN = 0x00001220;
static final long CKM_GOST28147_ECB = 0x00001221;
static final long CKM_GOST28147 = 0x00001222;
static final long CKM_GOST28147_MAC = 0x00001223;
static final long CKM_GOST28147_KEY_WRAP = 0x00001224;
static final long CKM_DSA_PARAMETER_GEN = 0x00002000;
static final long CKM_DH_PKCS_PARAMETER_GEN = 0x00002001;
static final long CKM_X9_42_DH_PKCS_PARAMETER_GEN =
    0x00002002;
static final long CKM_DSA_PROBABLISTIC_PARAMETER_GEN =
    0x00002003;
static final long CKM_DSA_SHAWE_TAYLOR_PARAMETER_GEN =
    0x00002004;
static final long CKM_AES_OFB = 0x00002104;
static final long CKM_AES_CFB64 = 0x00002105;
static final long CKM_AES_CFB8 = 0x00002106;
static final long CKM_AES_CFB128 = 0x00002107;
static final long CKM_AES_CFB1 = 0x00002108;
static final long CKM_AES_KEY_WRAP = 0x00002109;
static final long CKM_AES_KEY_WRAP_PAD = 0x0000210A;
static final long CKM_RSA_PKCS_TPM_1_1 = 0x00004001;
static final long CKM_RSA_PKCS_OAEP_TPM_1_1 = 0x00004002;

private static Map<Class<? extends PrivateKey>, Mechanism>
signingMechanisms;

static {
  signingMechanisms =
      new HashMap<Class<? extends PrivateKey>, Mechanism>();
  signingMechanisms.put(RSAPrivateKey.class,
      Mechanism.get(CKM_RSA_PKCS));
  signingMechanisms.put(ECDSAPrivateKey.class,
      Mechanism.get(CKM_ECDSA));
  signingMechanisms.put(DSAPrivateKey.class,
      Mechanism.get(CKM_DSA));
  signingMechanisms.put(KEAPrivateKey.class,
```

```
        Mechanism.get(CKM_DSA));
  }

  public static Mechanism getCorrespondentMechanism(
      PrivateKey privKey) {
    return signingMechanisms.get(privKey.getClass());
  }
}

package br.com.trapp.deviceserver.cryptographicmodule.
objects;

import javax.swing.GroupLayout;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.LayoutStyle;

public class Utils {

  public static char[] getPassword() {
    JPanel panel = new JPanel();
    GroupLayout layout = new GroupLayout(panel);
    panel.setLayout(layout);
    layout.setAutoCreateGaps(true);
    layout.setAutoCreateContainerGaps(true);
    JLabel explainText =
        new JLabel(
            "<html> <br>Se voc  deseja autorizar a "
            + "assinatura<br> insira o PIN do seu "
            + "dispositivo para que<br> a assinatura "
            + "se complete. <br>Sen o , pressione "
            + "Cancelar<html>",
            JLabel.CENTER);
    JLabel label = new JLabel("PIN:");
    JPasswordField pass = new JPasswordField(10);
    layout.setHorizontalGroup(layout
        .createSequentialGroup().addGroup(
            layout
                .createParallelGroup()
                .addComponent(explainText)
                .addGroup(
                    layout.createSequentialGroup()
                        .addComponent(label)
                        .addComponent(pass))));
    layout.setVerticalGroup(layout
        .createSequentialGroup()
        .addComponent(explainText)
        .addPreferredGap(
```

```
                LayoutStyle.ComponentPlacement.UNRELATED,
                GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(
                layout
                    .createParallelGroup(
                        GroupLayout.Alignment.LEADING)
                    .addComponent(label).addComponent(pass)));

        Icon icon = new ImageIcon("/home/marlon/icon.png");
        String[] options = new String[] { "OK", "Cancelar" };
        int option =
            JOptionPane.showOptionDialog(null, panel,
                "M dulo criptogr fico", JOptionPane.NO_OPTION,
                JOptionPane.PLAIN_MESSAGE, icon, options,
                options[0]);
        char[] password = null;
        if (option == 0) // pressing OK button
        {
          password = pass.getPassword();
        }
        return password;
      }

    }
```

```
package br.com.trapp.deviceserver.cryptographicmodule.
objects;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.lang.reflect.Field;
import java.security.MessageDigest;
import java.security.Security;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.xml.bind.DatatypeConverter;

import org.bouncycastle.asn1.ASN1ObjectIdentifier;
import org.bouncycastle.asn1.x509.AlgorithmIdentifier;
import org.bouncycastle.asn1.x509.DigestInfo;
import org.bouncycastle.cms.CMSSignedGenerator;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
```

```
import br.com.trapp.deviceserver.api.DeviceException;
import iaik.pkcs.pkcs11.Mechanism;
import iaik.pkcs.pkcs11.Module;
import iaik.pkcs.pkcs11.Session;
import iaik.pkcs.pkcs11.Slot;
import iaik.pkcs.pkcs11.Token;
import iaik.pkcs.pkcs11.TokenException;
import iaik.pkcs.pkcs11.TokenInfo;
import iaik.pkcs.pkcs11.objects.Attribute;
import iaik.pkcs.pkcs11.objects.Object;
import iaik.pkcs.pkcs11.objects.PrivateKey;
import iaik.pkcs.pkcs11.objects.PublicKey;
import iaik.pkcs.pkcs11.objects.X509PublicKeyCertificate;

public class CryptoDevices {

  private static List<String> MODULES;
  private static Map<CryptoDeviceInfo, Token> lastListingTokens;

  static {
    String os = System.getProperty("os.name").toLowerCase();
    if (os.contains("nux") || os.contains("nix"))
      MODULES =
          new ArrayList<String>(Arrays.asList(
              "libesp11_bio3k.so", "libepsng_p11.so",
              "libaetpkss.so", "libacospkcs11.so",
              "libASEP11.so", "libcastle.so.1.0.0.so",
              "libshuttle_p11v220.so", "opensc-pkcs11.so"));
    else if (os.contains("win"))
      MODULES =
          new ArrayList<String>(Arrays.asList("es1b3k.dll",
              "ep2pk11.dll", "aetpkss1.dll",
              "acospkcs11.dll", "asepkcs.dll",
              "ngp11v211.dll", "eps2003csp11.dll",
              "pronovacsp11.dll", "opensc-pkcs11.dll"));
    else if (os.contains("mac"))
      MODULES =
          new ArrayList<String>(Arrays.asList(
              "libepsng_p11.dylib", "libaetpkss.dylib",
              "libASEP11.dylib", "libcastle.1.0.0.dylib",
              "libacospkcs11.so"));

    Security.addProvider(new BouncyCastleProvider());
  }

  /**
   * List all cryptographic devices connected on this
   * computer, since any module can connect with it
   *
   * @return a list with all cryptographic devices connected
   *         on this computer
   */
```

```
public static Set<CryptoDeviceInfo> list() {
  Map<CryptoDeviceInfo, Token> tokens =
      new HashMap<CryptoDeviceInfo, Token>();

  for (int i = 0; i < MODULES.size(); i++) {
    Module pkcs11Module = null;

    try {
      pkcs11Module = Module.getInstance(MODULES.get(i));
      try {
        pkcs11Module.initialize(null);
      } catch (TokenException e) {
        // e.printStackTrace();
        // It might be opened previously, so we
        // ignore and keep
        // going
      }
      Slot[] slots =
          pkcs11Module
              .getSlotList(Module.SlotRequirement.
                  TOKEN_PRESENT);

      for (Slot slot : slots) {
        Token token = null;
        try {
          token = slot.getToken();
          token.openSession(
              Token.SessionType.SERIAL_SESSION,
              Token.SessionReadWriteBehavior.RO_SESSION,
              null, null);
          TokenInfo tokenInfo = token.getTokenInfo();
          CryptoDeviceInfo cryptoInfo =
              new CryptoDeviceInfo(
                  tokenInfo.getSerialNumber(),
                  tokenInfo.getLabel());
          Mechanism[] mechs = token.getMechanismList();
          for (Mechanism mechanism : mechs) {
            System.out.println(mechanism);
          }
          if (!tokens.containsKey(cryptoInfo)) {
            tokens.put(cryptoInfo, token);
          } else {
            // Select the compatible module
            // with more available
            // mechanisms.
            Mechanism[] oldMechanisms =
                tokens.get(cryptoInfo).getMechanismList();
            Mechanism[] newMechanisms =
                token.getMechanismList();
            if (oldMechanisms.length < newMechanisms.length)
              tokens.put(cryptoInfo, token);
          }
```

```
          } catch (Exception e) {
            // Error openning session, the
            // PKCS#11 module doesn't
            // support the card
          } finally {
            try {
              token.closeAllSessions();
            } catch (Exception e) {

            }
          }
        }
      } catch (IOException e) {
        // Error loading library, PKCS#11 module not
        // found, or it is not
        // a PKCS#11 module
      } catch (TokenException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
      }
    }
    lastListingTokens = tokens;
    return tokens.keySet();
  }

  /**
   * This method will return all certificates that have a
   * public key with the same ID of it, if the device
   * contains multiple certificates with the same ID it can
   * cause strange effects.
   *
   * @param tokenSerial
   *          the serial of the token that will receive this
   *          operation
   *
   * @return a list of ID and Label of each certificate that
   *         match the conditions
   */
  public static
      List<CertificateInfo>
      listCertsWithMatchingKey(CryptoDeviceInfo tokenSerial) {
    if (lastListingTokens == null)
      CryptoDevices.list();
    Token token = lastListingTokens.get(tokenSerial);
    Session session;
    List<CertificateInfo> matchedCerts =
        new ArrayList<CertificateInfo>();
    try {
      session =
          token.openSession(
              Token.SessionType.SERIAL_SESSION,
              Token.SessionReadWriteBehavior.RO_SESSION,
```

```
            null, null);

    Object[] objects;
    // FIND public keys
    List<PublicKey> publicKeys =
        new ArrayList<PublicKey>();
    session.findObjectsInit(new PublicKey());
    while ((objects = session.findObjects(10)).length > 0) {
      for (Object object : objects) {
        publicKeys.add((PublicKey) object);
      }
    }
    session.findObjectsFinal();

    // FIND certificates
    List<X509PublicKeyCertificate> certificates =
        new ArrayList<X509PublicKeyCertificate>();
    session
        .findObjectsInit(new X509PublicKeyCertificate());
    while ((objects = session.findObjects(10)).length > 0) {
      for (Object object : objects) {
        certificates
            .add((X509PublicKeyCertificate) object);
      }
    }
    session.findObjectsFinal();

    for (X509PublicKeyCertificate certificate : certificates) {
      for (PublicKey key : publicKeys) {
        if (key.getAttribute(Attribute.ID).equals(
            certificate.getAttribute(Attribute.ID))) {
          CertificateInfo certInfo =
              new CertificateInfo();
          certInfo.setId(certificate.getAttribute(
              Attribute.ID).toString());
          certInfo.setLabel(certificate.getAttribute(
              Attribute.LABEL).toString());
          certInfo.setDeviceSerial(tokenSerial
              .getSerial());
          matchedCerts.add(certInfo);
        }
      }
    }

  } catch (TokenException e) {
    e.printStackTrace();
  } finally {
    try {
      token.closeAllSessions();
    } catch (TokenException e) {
    }
  }
```

```
    return matchedCerts;
  }

  public static Certificate getCertificate(
      CryptoDeviceInfo tokenId,
      CertificateInfo certificateInfo) {
    if (lastListingTokens == null)
      CryptoDevices.list();
    Token token = lastListingTokens.get(tokenId);
    Session session;
    Certificate certificate = null;
    try {
      session =
          token.openSession(
              Token.SessionType.SERIAL_SESSION,
              Token.SessionReadWriteBehavior.RO_SESSION,
              null, null);

      Object[] objects;
      // FIND certificates
      CertificateTemplate template =
          new CertificateTemplate();
      if (certificateInfo.getId() != null)
        template.setIdAttribute(certificateInfo.getId());
      if (certificateInfo.getLabel() != null)
        template.setLabelAttribute(certificateInfo
            .getLabel());

      session.findObjectsInit(template);
      objects = session.findObjects(1);
      session.findObjectsFinal();
      if (objects.length > 0) {
        X509PublicKeyCertificate cert =
            (X509PublicKeyCertificate) objects[0];
        Attribute attr = cert.getAttribute(Attribute.VALUE);
        byte[] certBytes =
            DatatypeConverter.parseHexBinary(attr
                .toString());
        ByteArrayInputStream bais =
            new ByteArrayInputStream(certBytes);
        certificate =
            CertificateFactory.getInstance("x509")
                .generateCertificate(bais);
      }

    } catch (TokenException e) {
      e.printStackTrace();
    } catch (CertificateException e) {
    } finally {
      try {
        token.closeAllSessions();
      } catch (TokenException e) {
```

```
    }
  }
  return certificate;

}

public static byte[] sign(SignatureRequest signReq)
    throws DeviceException {
  if (lastListingTokens == null)
    CryptoDevices.list();
  Token token =
      lastListingTokens.get(new CryptoDeviceInfo(signReq
          .getTokenSerial(), null));
  Session session;
  try {
    session =
        token.openSession(
            Token.SessionType.SERIAL_SESSION,
            Token.SessionReadWriteBehavior.RW_SESSION,
            null, null);
    char[] psw = Utils.getPassword();
    if (psw != null) {
      session.login(Session.UserType.USER, psw);

      PrivateKeyTemplate templateSignatureKey =
          new PrivateKeyTemplate();
      templateSignatureKey.getSign().setBooleanValue(
          Boolean.TRUE);
      templateSignatureKey.setIdAttribute(signReq
          .getKeyId());

      byte[] bytes =
          Base64.getDecoder().decode(signReq.getData());
      session.findObjectsInit(templateSignatureKey);
      Object[] objects = session.findObjects(1);
      session.findObjectsFinal();
      if (objects.length > 0) {
        PrivateKey key = (PrivateKey) objects[0];
        MessageDigest digestEngine =
            MessageDigest.getInstance(
                signReq.getHashAlgorithm(), "BC");
        // String sha1Oid =
        // CMSSignedGenerator.DIGEST_SHA1;
        Field field =
            CMSSignedGenerator.class
                .getDeclaredField("DIGEST_"
                    + signReq.getHashAlgorithm());
        String hashOid = (String) field.get(null);

        // be sure that your token can process
        // the specified
        // mechanism
```

```
            Mechanism signatureMechanism =
                Mechanisms.getCorrespondentMechanism(key);
            if (signatureMechanism != null) {
              // initialize for signing
              session.signInit(signatureMechanism, key);

              byte[] digest = digestEngine.digest(bytes);
              DigestInfo di =
                  new DigestInfo(new AlgorithmIdentifier(
                      new ASN1ObjectIdentifier(hashOid)),
                      digest);
              byte[] signatureValue =
                  session.sign(di.getEncoded());
              session.closeSession();
              return signatureValue;
            } else {
              throw new DeviceException(
                  "Key type not supported");
            }
          }
        } else {
          throw new DeviceException(
              "The user doesn't allowed the signature");
        }
      } catch (Exception e) {
        throw new DeviceException(e.getMessage(), e);
      } finally {
        try {
          token.closeAllSessions();
        } catch (TokenException e) {
          // e.printStackTrace();
        }
      }
      return null;

    }
}

package br.com.trapp.deviceserver.cryptographicmodule.
objects;

import javax.xml.bind.DatatypeConverter;

import iaik.pkcs.pkcs11.objects.Attribute;
import iaik.pkcs.pkcs11.objects.ByteArrayAttribute;
import iaik.pkcs.pkcs11.objects.CharArrayAttribute;
import iaik.pkcs.pkcs11.objects.PrivateKey;

public class PrivateKeyTemplate extends PrivateKey {

  public PrivateKeyTemplate() {
    super();
```

```
  }

  @SuppressWarnings("unchecked")
  public void setIdAttribute(String value) {
    ByteArrayAttribute attr =
        new ByteArrayAttribute(Attribute.ID);
    attr.setByteArrayValue(DatatypeConverter
        .parseHexBinary(value));
    attributeTable_.put(Attribute.ID, attr);
  }

  @SuppressWarnings("unchecked")
  public void setLabelAttribute(String value) {
    CharArrayAttribute attr =
        new CharArrayAttribute(Attribute.LABEL);
    attr.setCharArrayValue(value.toCharArray());
    attributeTable_.put(Attribute.LABEL, attr);
  }
}
```

ANEXO E – Artigo SBC

Uma alternativa aos applets em java para acesso a dispositivos

Marlon Trapp[1]

[1]Instituto de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

Abstract. *With the descontinuation of the Java Applet plugin some web systems that used that plugin to access devices are without viable alternatives. In this work we study and look for an alternative to Java Applets for access devices in web applications. In the course of this work was defined a model that has the capacity of substitute the applets and offer access to the devices connected in the user's computer. The proposed model uses a modular design that allow connect another modules that can provide access to any kind of device. Finally this model was applied, this results in the development of a prototype called Device Server and a module to test the solution*

Resumo. *Com a descontinuação do plugin Java Applet alguns sistemas web que se utilizavam dessa tecnologia para acessar dispositivos ficaram sem alternativas viáveis para a sua substituição. Neste trabalho busca-se estudar e propor uma alternativa aos Java Applets para acesso a dispositivos em aplicações web. Ao decorrer do trabalho foi definido um modelo que tem a capacidade de substituir os applets e oferecer acesso aos dispositivos conectados no computador do usuário. O modelo proposto utiliza-se de uma modelagem modular a qual permite conectar outros módulos que possam prover acesso a qualquer tipo de dispositivo. Posteriormente esse modelo foi aplicado, o qual resultou no desenvolvido de um protótipo chamado de Device Server e um módulo com o intuito de testar a solução.*

1. Introdução

Com a popularização da internet a partir da metade dos anos 90 algumas empresas começaram a migrar para o novo ambiente virtual. No começo, se beneficiavam da possibilidade de digitalizar seus processos e informações e armazená-los remotamente, sendo acessíveis de qualquer lugar através da internet.

Com o passar do tempo e a expansão da internet as instituições e pessoas começaram a migrar tarefas cotidianas para o ambiente eletrônico. Um exemplo prático foi a criação do *internet banking*, o qual facilitou e agilizou processos que antes precisavam ser fisicamente executados dentro de uma agência. Mais recentemente ainda temos as lojas virtuais as quais tiveram um crescimento expressivo durante a última década.

Uma aplicação web é uma aplicação que é executada na internet através de um navegador. Ou seja, o processamento e armazenamento dos dados da aplicação ficam na web. Estas aplicações normalmente não precisam ser instaladas no computador do usuário. Alguns exemplos de aplicações web são clientes de e-mail e editores de arquivos online.

A aplicação web algumas vezes requer acesso a recursos da maquina onde está sendo executada, por exemplo um chat online que precisa acesso ao microfone e a câmera do computador. Nesse caso é facilmente atendida pois a tecnologia HTML5 e os navegadores suportam esse tipo de dispositivos. Porém quando a aplicação necessita acesso a um dispositivo menos usual, ou até proprietário, esse quadro se complica pois os navegadores não provêm esse suporte nativamente.

O acesso a dispositivos conectados ao computador do usuário através de navegadores usualmente pode ser feito de três maneiras. Primeira, por suporte do próprio navegador ao dispositivo, como acontece atualmente com webcams. Segunda, por extensões específicas para aquele navegador. E terceira, por meio de plugins de terceiros.

Dar suporte a uma classe de dispositivos utilizando a primeira solução é eficaz, porém precisa-se convencer a instituição desenvolvedora do navegador a fazê-lo. Além disso, essas instituições não irão se interessar em suportar um dispositivo que não é usado pela grande parte de seus usuários. Partimos então para a segunda opção, desenvolver uma extensão para o navegador. Essa alternativa também é interessante, porém necessitaria o desenvolvimento de uma extensão para cada combinação de SO e navegador, o que torna essa alternativa custosa para se desenvolver e manter. A terceira é a alternativa que se torna menos custosa, pois passa-se a responsabilidade da camada que executa no navegador para o plugin utilizado, e pode focar-se apenas em suportar diferentes SO's.

Um dos navegadores mais utilizados atualmente, o Google Chrome [NETMARKETSHARE 2016], está removendo o suporte a plugins implementados com a tecnologia *Netscape Plugin Application Programming Interface* (NPAPI) por questões de desempenho e segurança [Schuh 2014]. O NPAPI é uma *application programming interface* (API) que permite a criação de plugins para navegadores web. Ele foi criado em 1995 pela *Netscape Communications* e implementado primeiramente no navegador de mesmo nome, o Netscape, depois adotado por vários outros navegadores.

Um plugin utilizado para se fazer o acesso a dispositivos em navegadores é o Java Applet, ele carrega uma aplicação Java que interage com o navegador no momento em que o usuário acessa a página, tendo apenas as limitações da própria linguagem. Porém esse plugin utiliza a NPAPI como base e com a descontinuação do suporte à essa API ele não será mais suportado pelo navegador mais utilizado atualmente. Segundo informações publicadas no blog da Oracle o Java 9 não deve mais trazer o plugin junto à plataforma [Topic 2016]

1.1. Justificativa

No Laboratório de Segurança em Computação (LabSEC) da Universidade Federal de Santa Catarina (UFSC) utiliza-se um Java applet no projeto Hawa, um projeto de Autoridade Certificadora online. Esse applet é responsável por fazer a comunicação com dispositivos criptográficos conectados ao computador do usuário.

Com o anúncio da Oracle sobre a descontinuação do plugin, é interessante a análise de alguma alternativa para a substituição desse Java applet. E ainda, futuramente, possivelmente será necessário o o acesso a outros dispositivos como leitores biométricos.

1.2. Objetivos

Abaixo serão apresentados os objetivos deste trabalho.

1.2.1. Objetivo geral

O objetivo desse trabalho é fornecer uma alternativa ao plugin do Java Applet para acesso a dispositivos. Essa alternativa deve ter baixa dependência do navegador e da plataforma. Espera-se ainda que a solução seja segura, evitando ataques comuns para aplicações web como o *man-in-the-middle*, acesso não autorizado entre outras que possam prejudicar o usuário ou desenvolverdor.

1.2.2. Objetivos específicos

Como objetivos específicos espera-se:

1. Utilizar somente tecnologias já existentes em navegadores atuais para diminuir as dependências do projeto.
2. Minimizar as decisões necessárias por parte do usuário que muitas vezes não tem conhecimento técnico necessário para fazer configurações avançadas.
3. Deixar a solução livre de ataques do tipo man-in-the-middle, e prover sigilo em sua comunicação.

2. Proposta

No início do trabalho procurou-se uma forma de qualquer navegador se comunicar com algum artefato que tenha acesso aos dispositivos conectados ao computador do usuário, resolvemos analisar o que todos os navegadores têm em comum. Duas tecnologias básicas que estão em todos os navegadores são o suporte ao protocolo HTTP (*Hypertext Transfer Protocol*) e a scripts em JavaScript. Trabalhando a partir dessas duas tecnologias começou-se a criar o modelo de proposta.

2.1. Conexões

Com a tecnologia HTTP e JavaScript podemos executar quase tudo o que o navegador pode fazer, dentre elas acessar recursos em servidores, se esse servidor estiver na máquina do cliente então temos nosso artefato com acesso aos dispositivos conectados. Doravante chamaremos esse servidor de *Device Server*, ele estará instalado no computador do cliente, a arquitetura por parte do cliente pode ser vista na Fig. (1). A partir da comunicação do navegador com o dispositivo começou-se a analisar as alternativas para a comunicação com o aplicativo web que utilizará o dispositivo. O objetivo era que a solução seja igualmente, fácil de utilizar em um aplicativo web, e seja segura, não expondo os dados de dispositivo do usuário. Chegou-se a 2 modelos básicos de comunicação. O primeiro, mostrado na Fig. (2), consiste em uma comunicação direta do *Device Server* com o servidor web. Nesse modelo após o carregamento da página através da conexão 1, o navegador requisita a criação de uma sessão com o *Device Server*, pela conexão 2, informando o endereço do servidor web, assim os dois pontos se comunicam diretamente através da conexão 3. O segundo modelo, Fig. (3), não se comunica diretamente com o servidor web, toda a comunicação passa pelo navegador. Então, depois do carregamento da página pela conexão 1, o navegador requisita a criação de uma sessão com o *Device Server* através da conexão 2, o *Device Server* irá responder novamente pela conexão 2 ao navegador que encaminhará tudo para o servidor web através da conexão 1.

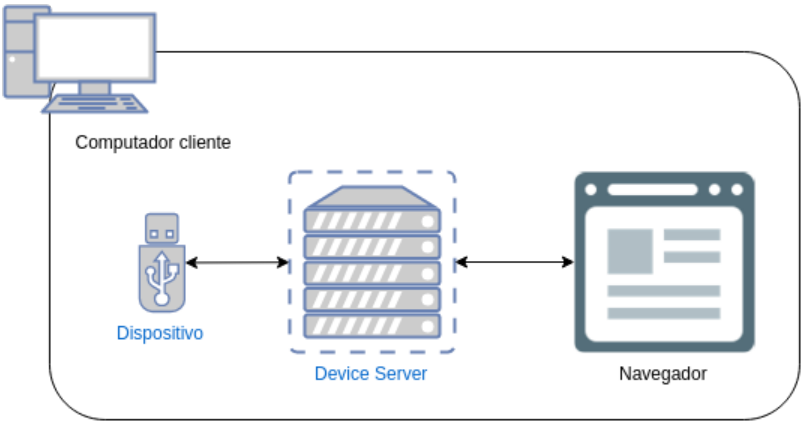

Figura 1. Modelo da solução por parte do cliente.

O primeiro modelo diminuiria a latência entre a comunicação do *Device Server* e o servidor web, porém abriría-se um vetor de ataque no computador do usuário, pois como o *Device Server* precisaria ser acessado diretamente da rede externa agentes maliciosos poderiam se utilizar disso para explorar alguma possível brecha. A comunicação entre o *Device Server* e o servidor web teria que ser padronizada, e a aplicação que quisesse utiliza-lo teria que respeitar esse padrão. Já o segundo modelo precisará de uma intervenção do navegador para cada comunicação feita com o *Device Server*. Apesar do custo, isso deixará a solução menos passível de ataques, pois o *Device Server* receberá requisições somente no localhost. Na segunda solução, como o navegador intermedeia a conexão, fica a cargo dos desenvolvedores da aplicação escolher como serão repassado as informações ao servidor web, por AJAX (*Asynchronous Javascript and XML*), REST (*Representational State Transfer*), SOAP (*Simple Object Access Protocol*), etc. Como o foco primário da solução é segurança decidimos por utilizar o segundo modelo, com conexão indireta.

2.2. Protocolos e padrões de comunicação

Definido o modelo de conexões, passou-se à escolha de como cada ponta se comunicará com a outra. Com base na Fig. (3), a conexão 1 fica a critério do desenvolvedor da aplicação web, já a conexão 2, entre o navegador e o *Device Server* precisa ser especificada.

Levantou-se duas alternativas, *websockets* ou REST. Para a solução de acesso a dispositivos a alternativa mais interessante seria a utilização de *websockets*, pois ela permite comunicação assíncrona e ainda mais rápida do que com REST, porém como é uma especificação recente e somente os navegadores mais atualizados a suportam optou-se pela utilização do REST. O padrão REST já está bem estabelecido e já existem bibliotecas que facilitam muito sua implementação.

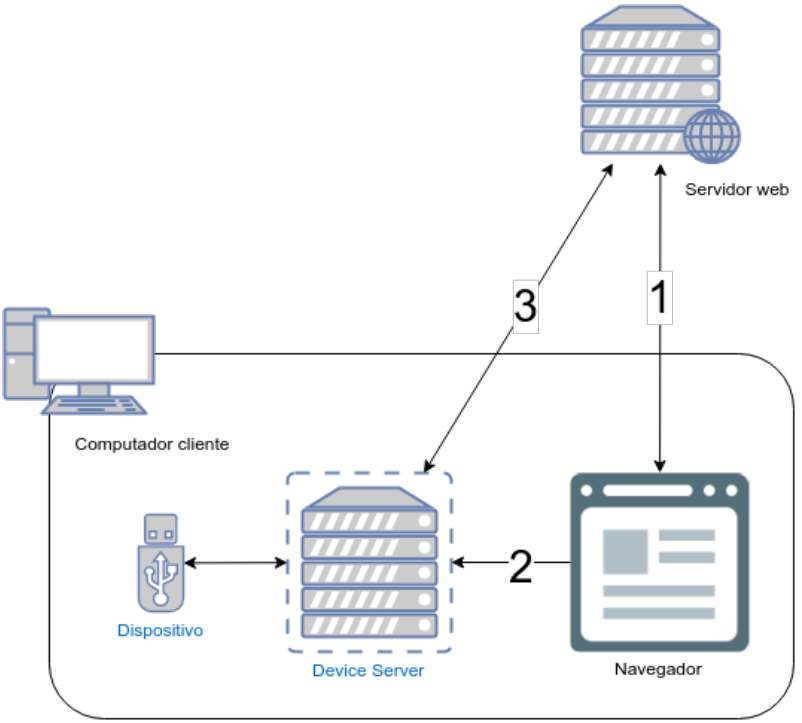

Figura 2. Modelo de conexão direta.

2.3. Seguraça da solução

Como o projeto interage com dispositivos do usuário os quais podem conter informações sensíveis, como dispositivos criptográficos, focou-se na questão segurança. O projeto se preocupa tanto em autenticar quem está requisitando acesso ao dispositivo quanto a como os dados devem trafegar até o servidor web.

2.3.1. Autenticação

A autenticação é feita através de certificados digitais, ou seja, teremos uma AC (autoridade certificadora) que emitirá certificados às entidades ou aplicações que desejam acesso ao *Device Server*. Com isso toda a aplicação terá acesso a um par de chaves atrelado a um certificado o qual será usado para fazer sua autenticação. Conforme a Fig. (4) podemos ver como funcionará na prática o esquema de autenticação. No passo 1, quando o navegador requisita a página, o servidor web retorna-a e junto envia o certificado para acesso ao *Device Server*. Assim a página carregada no navegador poderá requisitar a abertura de uma sessão com o *Device Server*, mostrada no passo 2, passando o certificado da aplicação

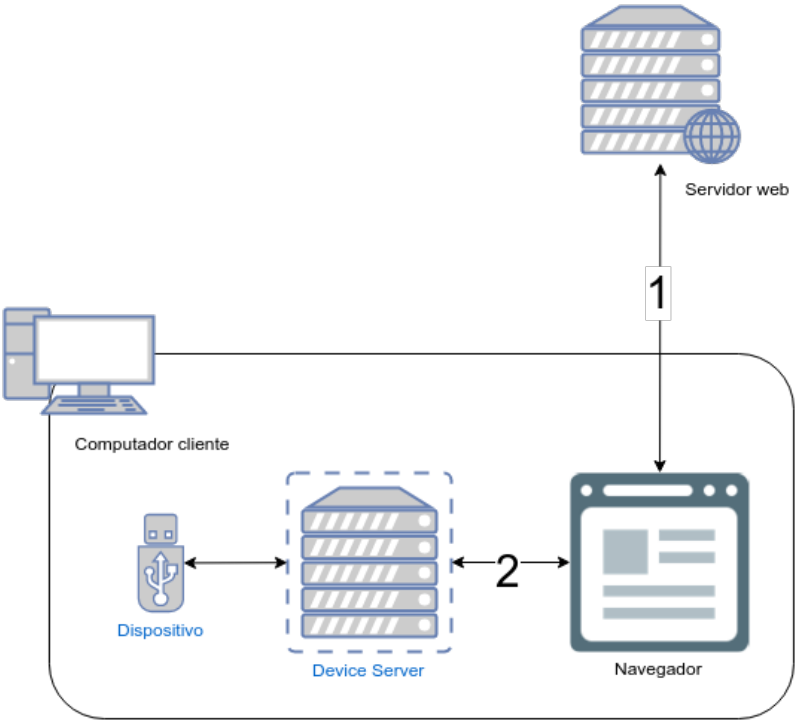

Figura 3. Modelo de conexão indireta.

e o módulo a qual deseja acesso. O *Device Server* verifica se o certificado é valido e se não está revogado através de uma LCR (Lista de certificados revogados). Passando nessas duas verificações o *Device Server* começa a execução do protocolo TLS, assim gerando um canal de comunicação seguro entre ele e o servidor web, passo 3 na Fig. (4). Estabelecido o protocolo o *Device Server* liberará o acesso ao módulo requisitado através dessa sessão.

ACAutoridade Certificadora

2.3.2. Sigilo

O sigilo é alcançado cifrando as mensagens passadas de uma ponta a outra, podería-se utilizar as chaves atreladas ao certificado usado no passo de autenticação para cifrar e decifrar mensagens, porém como sabemos [Schneier 1996, 10, p. 216] o uso de criptografia assimétrica para cifragem de decifragem é menos eficiente que o uso de criptografia simétrica. Por isso escolhemos, ao invés de usar as chaves dos certificados gerados para a autenticação, executar o protocolo TLS (*Transport Layer Security*) usado na internet

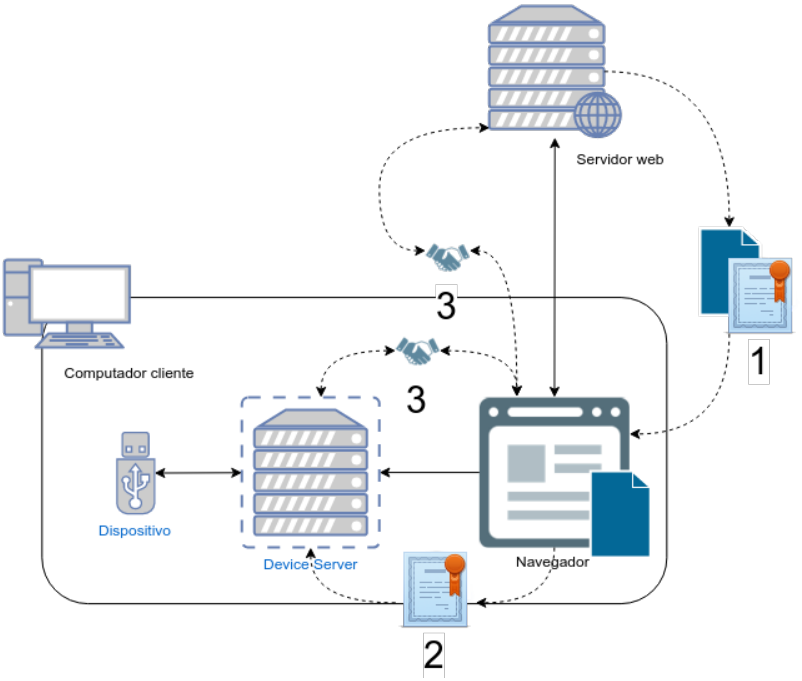

Figura 4. Esquema de autenticação.

para garantir que o site que está sendo acessado é realmente quem diz ser. Então, usando novamente a Fig. (4), ao final do passo 3, o servidor web e o *Device Server* terão uma conexão por um tunel TLS. Assim podemos usar essa conexão sem nos preocuparmos se o meio está comprometido, por exemplo, se houver uma brecha no navegador que possibilite que uma página acesse o conteúdo de outra aberta ao mesmo tempo. Com o uso do TLS impossibilita-se o ataque, pois mesmo que a página maliciosa consiga saber qual a sessão, ela não conseguira transmitir mensagens que façam sentido quando o *Device Server* decifrá-las com a chave de gerada pelo protocolo TLS. Dessa maneira estará evitando-se os ataque do tipo *man-in-the-middle*.

3. Protótipo

O protótipo foi desenvolvido com a linguagem de programação Java, para alcançar múltiplas plataformas sem a necessidade de reescrita de código. Além do protótipo do *Device Server* foram desenvolvidos um módulo de teste para ser gerenciado pelo *Device Server* e uma pequena aplicação web, escrita totalmente em JavaScript e HTML, como prova de conceito.

3.1. Device Server

O desenvolvimento começou pela ideia base da proposta, a criação de um servidor local para a comunicação com o navegador. Para isso se tornar mais fácil procurou-se por implementações de servidores web em Java que suportassem a adição do Java API for RESTful Web Services (JAX-RS), especificada pela JSR 339, a qual ajuda o desenvolvimento de serviços REST em java usando anotações para simplificar o processo. Dentre as várias opções de projetos analisados chegou-se ao Jetty e ao Grizzle. Ambos utilizam o Jersey para prover as funcionalidades do JAX-RS. Após o teste das soluções, notou-se que ambas solucionavam o problema de criar um servidor local, porém por questões de segurança decidiu-se utilizar o Grizzle pela possibilidade da configuração para ouvir somente requisições vindas de localhost, enquanto o Jetty não oferecia essa opção, tendo que implementar filtros para recusar conexões externas.

Uma simplificação do modelo que foi aplicada no protótipo foi a troca do protocolo TLS pela execução do protocolo de *Diffie-Hellman*. O qual tornou a implementação mais simples, mas ainda assegurando certa segurança a implementação.

Para criar o *Device Server* analisou-se os recursos necessários para o seu funcionamento, os recursos que foram definidos como necessários estão listados na Tab. (1).

Tabela 1. Tabela de recursos

	Método	URL
1	GET	http://localhost:9977/deviceserver/
2	POST	http://localhost:9977/deviceserver/startsession /{module}
3	POST	http://localhost:9977/deviceserver/establishsession /{session}
4	POST	http://localhost:9977/deviceserver/{session}

O primeiro método, método GET na raiz do recurso do *Device Server*, lista todos os módulos carregados no *Device Server*. Para estabelecimento da sessão criou-se dois métodos, o primeiro é um método POST, o *startsession*, recurso 2 na Tab. (1), que tem como parâmetro da URL o módulo que se deseja acesso, além disso deve-se passar o certificado da aplicação cofdificado em Base64 para conferir se ele tem acesso ao *Device Server*.

Chegando ao *Device Server* ele checa se o módulo desejado existe e está carregado, depois verifica a validade do certificado enviado junto a requisição de inicialização da sessão. Tudo estando correto ele gera os parâmetros do protocolo de Diffie-Hellman e responde à aplicação.

A aplicação em posse dessa resposta gera a sua parte dos parâmetros do protocolo Diffie-Hellman, assina-os com a chave correspondente ao seu certificado e envia os parâmetros e a assinatura para o segundo método para o estabelecimento da sessão, o *establishsession*, recurso 3 da Tab. (1).

Agora o *Device Server* pode verificar a assinatura dos parâmetros e assim, se estive

correta, autorizar a inicialização da sessão. A partir daí a aplicação pode se comunicar com o módulo através do método 4 da Tab. (1).

3.1.1. Módulos

O *Device Server* foi pensado para ser o meio para que aplicações web e dispositivos se comuniquem com segurança e facilidade, pensando nisso resolveu-se não amarrar nenhuma implementação de acesso a dispositivo com o *Device Server* em si, resolvemos usar a abordagem de programação orientada a componentes, onde os módulos possam ser encaixados no *Device Server* e assim serem acessados com a intermediação dele, o diagrama de componentes é mostrado na Fig. (5).

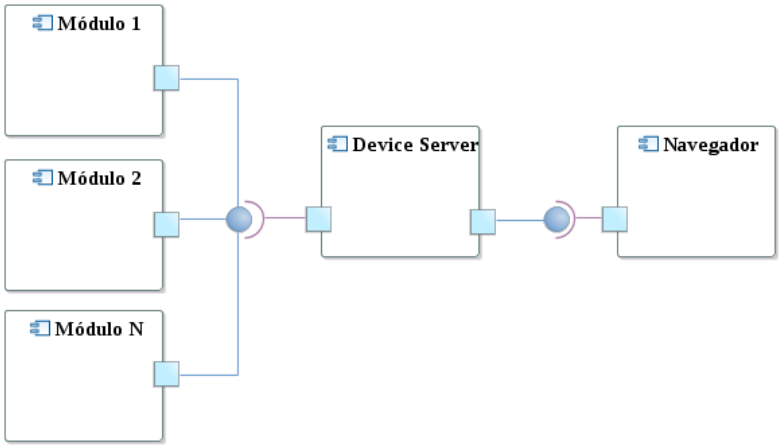

Figura 5. Diagrama de componentes.

O carregamento dos módulos é feito na inicialização do *Device Server*, que procura dentro da pasta onde foi executado o diretório *modules* e carrega todos os arquivos ".jar"que estão nessa pasta e respeitam a interface previamente definida.

3.2. Echo Module

Para testarmos a solução foi implementado um módulo simples para o *Device Server*, ele possui dois métodos, o método *echo* que retorna a mesma mensagem que a aplicação lhe enviou e o método *echoreverse* que retorna o reverso da mensagem enviada a ele. Além disso desenvolvemos uma pagina em HTML e alguns scripts em Javascript para se comunicar com o *Device Server* e verificar o correto funcionamento. Num cenário real o uso do *Device Server* por uma aplicativo Javascript é desaconselhável por expõe a chave privada do certificado da aplicação.

Com a aplicação de teste além da verificação do correto funcionamento do *Device Server*, medimos os tempos para a estabilização da sessão e das consecutivas

comunicações afim de verificar qual é o custo do uso dessa solução. Como o módulo *echo* é extremamente simples desconsiderou-se o tempo de processamento dele pois é irrisório comparado com o tempo do processamento da requisição, o medição foi feita executando cada teste cinco vezes para tirarmos uma média. na Tab. (2) podemos ver os tempos (em milissegundos) que cada repetição levou para estabelecer uma sessão com o módulo. E na Tab. (3) os tempos entre o envio e o recebimento de uma mensagem ao módulo.

Tabela 2. Tempos para estabelecimento de sessão

Experimento	Tempo
1	718 ms
2	870 ms
3	903 ms
4	881 ms
5	872 ms
Média	848,8 ms

Tabela 3. Tempos de comunicação

Experimento	Tempo
1	11 ms
2	10 ms
3	8 ms
4	10 ms
5	9 ms
Média	9,6 ms

3.3. Cryptographic Module

Visando um resultado mais concreto também desenvolvemos um módulo capaz de se comunicar com dispositivos criptográficos. Esse módulo consegue listar dispositivos compatíveis, listar certificados e realizar assinatura digital. Os métodos implementados são:

Tabela 4. Métodos Cryptographic Module

Método	Descrição
list	Lista os dispositivos compativeis conectados
listCerts	Lista os certificados dos dispositivos escolhidos
sign	Assina uma mensagem com a chave escolhida

Para testar esse módulo desenvolvemos também uma nova aplicação web desenvolvida com Javascript e HTML.

4. Conclusão

Nesse trabalho foi apresentado o conceito de aplicações web e sua necessidade de, em alguns casos, ter acesso a dispositivos conectados no computador do usuário. Uma maneira

de se acessar dispositivos através do navegador é o uso do plugin Java Applet. Porém, o navegador mais utilizado atualmente desativou o suporte a esse plugin. Esse trabalho propôs uma alternativa a esse plugin, chamado de *Device Server*.

No capítulo 2 mostrou-se como é possível desacoplar o *Device Server* de um navegador específico usando requisições HTML e Javascript, duas tecnologias presentes em todos os navegadores modernos. Propomos ainda, a independência de plataforma utilizando a linguagem de programação Java, a qual pode ser executada na maioria dos SOs atuais.

Propomos também, para resolver o problema de acesso a diferentes tipos de dispositivos, usar uma abordagem modular. Cada módulo é dedicado a uma classe ou tipo de dispositivo. Isso deixou a solução mais genérica permitindo o desenvolvimento posterior de módulos para dispositivos não planejados inicialmente.

Na seção 2.3 mostrou-se como a utilização de certificados digitais pode ser usada para autenticar a aplicação que deseja acesso à nossa solução. Isso garante o acesso somente para quem deveria poder acessar os dispositivos. Ainda na questão de segurança, alcançou-se o sigilo da comunicação entre o *Device Server* e a aplicação web através o protocolo TLS, o qual no protótipo foi simplificado e utilizado a troca de chaves de Diffie Hellman. Esses dois artefatos nos garantem autenticidade e sigilo, aumentando a segurança da solução.

No capítulo 3 desenvolvemos uma prova de conceito do modelo proposto implementando todas as funcionalidades propostas no capítulo 2. Ainda nesse capítulo, na seção 3.2, desenvolvemos um módulo para acoplarmos ao *Device Server* e testar seu funcionamento. O qual nos deu um resultado preliminar quanto à performance da solução proposta. Em sequencia, na seção 3.3, foi apresentado outro módulo que faz a comunicação com dispositivos criptográficos reais conectados no computador do usuário.

5. Referências

Referências

Adams, C. and Lloyd, S. (2003). *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley, S.l.

Housley, R., Laboratories, R., Polk, W., NIST, Ford, W., VeriSign, Solo, D., and Citigroup (2002). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.

Housley, R. and Polk, T. (2001). *Planning for PKI: Best Practices Guide for Deploting Public Key Infrastructure*. John Wiley & Sons, Inc, S.l.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. (2001). *Handbook of Applied Cryptography*. CRC Press, S.l., 5 edition.

NETMARKETSHARE (2016). Desktop Browser Market Share.

Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc, New York, N.Y.

Schuh, J. (2014). The Final Countdown for NPAPI.

Stallings, W. (2003). *Cryptography and Network Security: Principles and Practices*. Pearson Education, Inc., Upper Saddle River, N.J.

Topic, D. (2016). Moving to a Plugin-Free Web.

Williams, M. Java se 8: Creating a basic rest web service using grizzly, jersey, and maven.