UNIVERSIDADE FEDERAL DE SANTA CATARINA

Uma alternativa aos Applets em Java para realização de assinaturas digitais através do navegador web

David Grechi Doll

Florianópolis

2018/1

David Grechi Doll

Uma alternativa aos Applets em Java para realização de assinaturas digitais através do navegador web

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do título de Bacharel, do curso de Sistemas de Informação na Universidade Federal de Santa Catarina.
Orientador: Profª Carla Merkle Westphall, Dr.

Florianópolis

2018/1

David Grechi Doll

Uma alternativa aos Applets em Java para realização de assinaturas digitais através do navegador web

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do título de Bacharel, do curso de Sistemas de Informação na Universidade Federal de Santa Catarina.

Profº Cristian Koliver, Dr.
Coordenador do Curso

Banca Examinadora:

Profª Carla Merkle Westphall, Dr.
Orientador
Universidade Federal de Santa Catarina

Profº Joao Bosco Mangueira Sobral, Dr.
Universidade Federal de Santa Catarina

Profº Jean Everson Martina, Dr.
Universidade Federal de Santa Catarina

Alex Sandro da Silva Pereira
Universidade Federal de Santa Catarina

Florianópolis
2018/1

Resumo

Por muitos anos utilizamos uma assinatura para confirmar a validade e manifestar o acordo à determinada documentação, atualmente a assinatura manuscrita vem sendo substituída por assinaturas digitais. Baseada na criptografia de chaves públicas, a assinatura digital tem como objetivo assinar documentos eletrônicos, garantindo a mesma equivalência jurídica que as assinaturas realizadas no papel. Desde então, diversos softwares que realizam assinaturas digitais vêm sendo desenvolvidos, e com o crescente uso das aplicações Web e o abandono de aplicações desktop, torna-se cada vez mais necessário o desenvolvimento de tecnologias que possibilitem a assinatura através do navegador. Com a descontinuação do plugin Java Applet, alguns sistemas Web que o utilizavam para gerar assinaturas, tiveram que buscar alternativas para a sua substituição. Este trabalho tem como objetivo apresentar soluções para gerar assinaturas através do navegador Web, além de um protótipo funcional que solucione este problema da forma mais adequada.

Palavras-chave: Assinatura digital, Criptografia de chaves públicas, NPAPI, Applet, Web Extensions, Native Messaging.

Abstract

For many years we have used a signature to confirm the validity and express the agreement to certain documentation, currently the handwritten signature has been replaced by digital signatures. Based on the public key cryptography, the digital signature aims to sign electronic documents, guaranteeing the same legal equivalence as the signatures made on paper. Since then, several softwares that have made digital signatures have been developed, and with the increasing use of web browsers and the abandonment of desktop applications, it is becoming increasingly necessary to develop technologies that enable the signature through the browser. With the discontinuation of the Java Applet plugin, some web systems that used it to make signatures, had to look for alternatives for its replacement. This work aims to present solutions to generate signatures through the browser, as well as a functional prototype that solves this problem in the most appropriate way.

Keywords: Digital signature, Public Key Cryptography, NPAPI, Applet. Applet, Web Extensions, Native Messaging..

Lista de ilustrações

Lista de abreviaturas e siglas

API	*Application Programming Interface*
JSON	*JavaScript Object Notation*
TCC	*Trabalho de Conclusão de Curso*
CMS	*Cryptographic Message Syntax*
IETF	*Internet Engineering Task Force*
HTTP	*Hypertext Transfer Protocol*

Sumário

1 Introdução

O século XXI é marcado pelo predomínio do conhecimento e da informação, a Internet nos proporciona maior eficiência e rapidez na transmissão de dados e produção. Através do seu acesso universal, barreiras territoriais são desfeitas, permitindo que pessoas se reúnam em torno de ideias em lugares virtuais (GANDINI; SALOMÃO; JACOB, 2001).

Os negócios eletrônicos feitos através da Internet já somam bilhões de dólares, e na mesma proporção que crescem, aumentam as fraudes de identidade. Uma das soluções para esta insegurança virtual pode ser o uso da assinatura digital (FERREIRA, 2001).

A assinatura digital é utilizada para agregar confiança e segurança às comunicações e negócios vinculados a um ambiente virtual como a Internet, oferecendo eficiência e rapidez. Além disso, a assinatura digital contribui de forma positiva para o meio ambiente, empresas que armazenam milhares de documentos poderiam digitalizar os mesmos, garantindo a sua validade jurídica através de assinaturas digitais (MENKE, 2003).

A assinatura digital é baseada na criptografia assimétrica, que consiste no uso de duas chaves, pública e privada, uma utilizada pelo remetente e outra pelo receptor da mensagem, e é sobre este conceito que se baseia uma assinatura digital. Este par de chaves é gerado por programas de computador ou hardware e as chaves atuam em conjunto. Tudo que é cifrado pela chave pública, só pode ser decifrado pela chave privada correspondente e vice versa. Uma assinatura digital é feita através do uso da chave privada e a verificação é com o uso da chave pública. Somente a chave pública correspondente aquela chave privada poderá interpretar corretamente a assinatura (STALLINGS, 2008).

Para agregar mais segurança ao processo de assinatura, é necessário garantir que a chave privada utilizada para assinar pertence ao assinante. Um certificado digital é o elemento que garante esta autenticidade, porque um certificado é assinado por uma terceira parte confiável, a qual comprova o vínculo entre o assinante e sua chave pública. Na prática, o certificado digital, que contém a chave pública do assinante, é enviado juntamente com a assinatura, desta forma é possível verificar a assinatura e atestar a autenticidade bem como a validade desta informação.

Um dos aspectos de segurança do processo de geração da assinatura digital consiste no armazenamento da chave privada num local que seja acessível somente pelo assinante. Ou

seja, a chave privada deve ser armazenada num local privado. Usualmente, a chave privada fica armazenada em algum dispositivo criptográfico, como SmartCards ou tokens. Portanto para realizar uma assinatura digital através de um navegador é necessária uma solução que permita o acesso a chave armazenada nestes dispositivos. De acordo com (REIS; BARTH; PIZANO, 2009) navegadores como Google Chrome e Mozilla Firefox funcionam em um modo conhecido como sandbox, um mecanismo de segurança que impede que chamadas nativas sejam feitas ao sistema operacional, tornando inviável o acesso direto a estes dispositivos bem como o acesso a chave privada para realizar assinaturas.

As tecnologias de acesso a chaves criptográficas e outras funções de segurança através de navegadores sempre foram um tópico desafiador. Em um primeiro momento as formas de acessar estas funções eram apenas através de plugins. O navegador deve isolar o ambiente das páginas acessadas do computador por questões de segurança. Podemos separar os plugins feitos para browsers em dois grandes grupos: os especializados e os de uso genérico.

Como exemplos de plugins de uso genérico podemos citar o Flash (ADOBE, 2017) e o Java (SMITH, 2017). Exemplos de plugins de uso especializado temos por exemplo o plugin para o skype web (SKYPE, 2017).

A primeira arquitetura de plugins utilizada para browsers surgiu com o Netscape. Conhecida como Netscape Plugin API (NPAPI), essa arquitetura foi adotada pela maioria dos browsers dominantes do mercado entre 1995 e 2013. Entretanto, devido a problemas recorrentes de segurança em plugins desenvolvidos com essa API (Oracle, 2016), seu uso foi descontinuado (SMEDBERG, 2015; Sylvain, Nicolas, 2014).

Esta tecnologia possibilita a utilização do plugin Java Applet, que é utilizado para realizar assinaturas através de um navegador. Devido à descontinuação da mesma, este TCC tem como objetivo apresentar soluções alternativas para a solução deste problema, através de um protótipo funcional.

1.1 Objetivos

Nesta seção são apresentados o objetivo geral e os objetivos específicos deste trabalho.

1.1.1 Geral

O objetivo deste trabalho é fornecer uma alternativa para gerar assinaturas digitais através de um navegador Web, já que a tecnologia Java Applet não é mais suportada.

1.1.2 Específicos

Os objetivos específicos deste trabalho que podem ser listados são:

- Encontrar alternativa atual que solucione este problema;
- Desenvolver um protótipo capaz de gerar uma assinatura CMS (Cryptographic Message Syntax) através de um navegador Web;
- Evitar a instalação de dependências externas, como por exemplo o Java;
- Deixar a solução livre de conflitos, como anti-vírus e firewall, sem que se torne vulnerável a ataques;
- Não solicitar permissão de administrador para que a aplicação seja instalada.

1.2 Justificativa

Na BRy Tecnologia, empresa de tecnologia de Florianópolis, utiliza-se Java applets para realizar assinaturas através do navegador. Devido à descontinuação da tecnologia que possibilita o seu uso, tornou-se neecessário o desenvolvimento de uma solução alternativa. Em um prmeiro momento, foi desenvolvida uma solução que utilizava websockets, uma tecnologia avançada que torna possível abrir uma sessão de comunicação interativa entre o navegador do usuário e um servidor (FLR, 2017). Porém, esta solução se mostrou inadequada, já que ela depende da instalação de uma máquina virtual Java no computador do cliente. Além disso, diversos problemas relacionados a Antivírus foram relatados, e também a necessidade de permissão de administrador para realizar a sua instalação. Dadas as justificativas, é necessário realizar a busca por uma nova alternativa, que resolva este problema, e que seja livre dos problemas mencionados anteriormente.

2 Fundamentação Teórica

Este capítulo apresenta uma descrição dos conceitos essenciais para o entendimento e desenvolvimento deste trabalho.

2.1 Criptografia

Criptografia é a escrita de forma ilegível. Cripto, do grego “kryptos”, significa escondido, oculto, e grafia, também do grego “graphos”, significa escrita (PEREIRA, 2012).

O aumento do risco de roubo de informações peculiares, cresceu com o uso do computador e dos sistemas de comunicação pela indústria. O aspecto mais importante, na segurança da informação, hoje, é a criptografia, como um componente básico de um computador. De acordo com (STALLINGS, 2008), a criptografia é a ferramenta mais importante para a segurança da rede e das comunicações e normalmente é utilizada de duas formas, a criptografia simétrica ou assimétrica.

Com a necessidade de transmitir informações de maneira sigilosa, surgiu este conceito, que consiste no simples embaralhamento das letras, ou até operações matemáticas (SCHNEIER, 2007). A cifragem é uma aplicação muito conhecida da criptografia, atualmente, dados e informações são representados de forma binária, e através de funções criptográficas de cifragem, a criptografia permite transformar dados legíveis em conteúdos ilegíveis, chamados conteúdos cifrados.

Alguns conceitos relacionados diretamente a criptografia são importantes para que seja possível o entendimento do que ainda será apresentado sobre este tema:

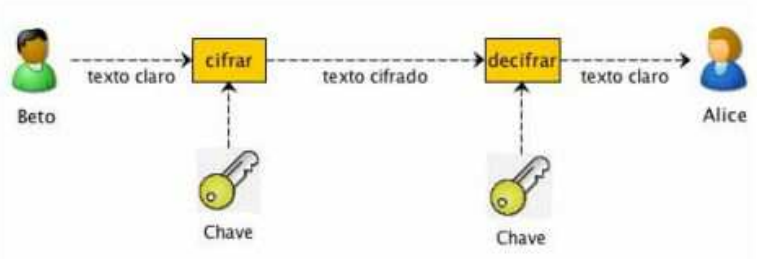
- *Texto claro:* Texto original, não cifrado;
- *Texto cifrado:* Texto ilegível, não compreensível;
- *Cifrar:* Transformar texto claro em texto cifrado;
- *Decifrar:* Transformar texto cifrado em texto plano;
- *Chave:* Conjunto de dados utilizados para cifrar;

- *Confidencialidade:* Um dado pode ser visualizado somente por aqueles que possuem autorização;
- *Integridade:* Garantir que um dado não foi modificado;
- *Não-Repúdio:* Uma ação ou autorização realizada, como por exemplo uma assinatura, não pode ser recusada;
- *Autenticidade:* Garantia de que uma entidade é legítima, ou seja, é quem diz ser. Por exemplo, garantir que uma assinatura foi realmente gerada por uma entidade X.

2.1.1 Criptografia Simétrica

A criptografia simétrica, chamada de criptografia convencional ou até criptografia de chave única, é um dos tipos mais utilizados de criptografia (STALLINGS, 2008). Na criptografia simétrica, uma mensagem é cifrada e decifrada através da mesma chave, chamada de chave secreta. Ou seja, a chave utilizada para cifrar será utilizada também para decifrar. Neste caso, o emissor e o receptor da mensagem compartilham a mesma chave secreta .

Figura 1 – Criptograia Simétrica



Fonte: Sutil et al. (2010, p. 11)

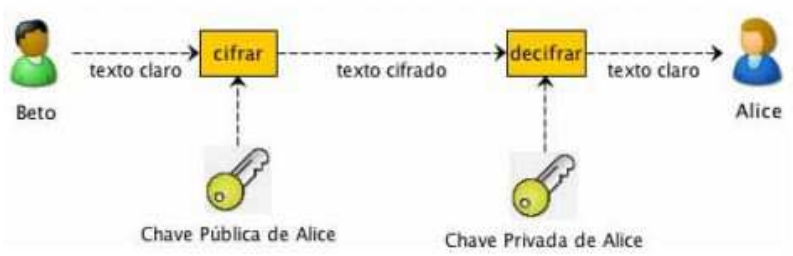
Como pode ser visto na figura 1, Alice e Beto compartilham uma chave secreta, e a utilizam para cifrar e decifrar o texto, neste caso, enquanto somente Beto e Alice tiverem acesso a esta chave, ninguém pode ler o conteúdo do texto claro.

Para utilizar a criptografia simétrica é necessário levar em conta a forma como estas chaves serão destribuidas de forma segura, se a mensagem foi modificada, e também se uma terceira parte conseguiu acesso a uma chave, além disso, é importante também, reconhecer se uma mensagem foi realmente cifrada pela pessoa que diz ter cifrado, por exemplo, se alguém obtém acesso a chave de Beto, é possível que Alice decifre uma mensagem e acredite que foi Beto quem enviou.

2.1.2 Criptografia Assimétrica

Para responder as questões levantadas pelo uso da criptografia simétrica, surgiu a criptografia assimétrica, publicada em 1976 por *Diffie Hellman* (ALFRED; PAUL; SCOTT, 1996). A principal diferença entre elas, é que a criptografia assimétrica utiliza um par de chaves para cifrar e decifrar, ou seja, cada chave é utilizada para um propósito diferente (SUTIL et al., 2010).

Figura 2 – Criptograia Assimétrica



Fonte: Sutil et al. (2010, p. 11)

Na figura 2, podemos ver que Beto utiliza a chave pública de Alice para cifrar o texto, e Alice decifra o texto cifrado por Beto com a sua chave privada. O conceito de criptografia assimétrica envolve um par de chaves, pública e privada. A chave privada fica sempre em posse do seu dono, e nunca deve ser compartilhada, já a chave pública é distribuida livremente. Estas chaves são complementares, ou seja, tudo que é cifrado com uma, deve ser decifrado com a outra e vice versa.

Utilizando este conceito, Beto pode cifrar uma mensagem utilizando a chave pública de Alice, e somente Alice poderá decifrar já que só ela possui a chave privada. É possível também que Alice cifre uma mensagem através de sua chave privada, e qualquer pessoa que decifrar a mesma utilizando a chave pública de Alice, terá certeza que foi ela quem cifrou, porque somente Alice tem acesso à sua chave privada. Estas duas abordagens, garantem respectivamente, o sigilo e autenticidade da mensagem.

2.1.3 Funções resumo (HASH)

Funções hash de resumo criptográfico tem um papel fundamental na criptografia. Funções hash recebem uma mensagem como entrada e transformam em uma saída conhecida como hash (ALFRED; PAUL; SCOTT, 1996). Estas funções resumo, também conhecidas como

funções hash, são uma operação matemática, que transforma um conjunto de dados em uma sequência de dados de tamanho fixo, geralmente menor do que a sequência original (SUTIL et al., 2010).

Essa sequência busca identificar um arquivo ou dado unicamente. O resumo criptográfico de um dado, se utilizada a mesma função, será sempre o mesmo, ou seja, se um dado for modificado, o seu resumo criptográfico também será, garantindo assim a sua integridade. Funções resumo não permitem o processo inverso, deve ser impossível obter o conteúdo original utilizado para gerar um hash (STALLINGS, 2008). Além disso, segundo (SUTIL et al., 2010), como o tamanho do hash é limitado, podem existir colisões, resumos iguais para entradas diferentes, porém, quanto maior o hash menor a chance de colisão.

Figura 3 – Função Hash

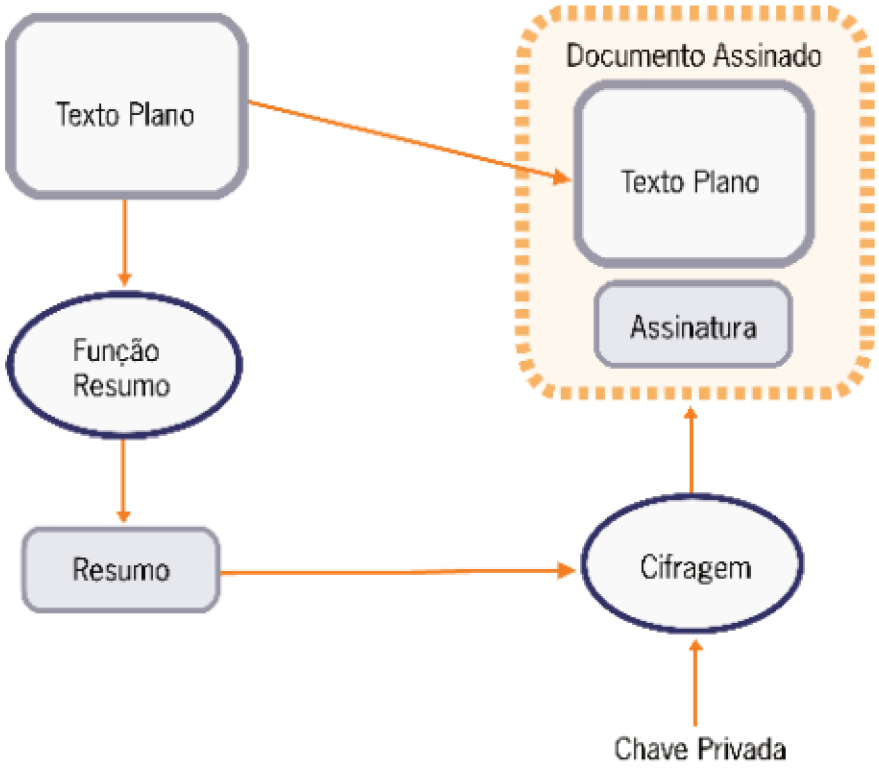
Fonte: Eltiger (2008)

2.1.4 Assinatura digital

Baseada na criptografia de chaves públicas, a assinatura digital permite assinar um documento eletrônico com a mesma validade jurídica de um documento assinado no papel, de forma muito mais segura e promovendo a eliminação de tempo e custos com impressão, transporte, armazenamento e despesas cartorárias.

A assinatura digital provê a garantia de integridade, autenticidade e não repúdio de uma mensagem. Como pode ser visto na figura 4, para gerar uma assinatura, deve-se utilizar uma função de resumo para obter o hash do texto plano, e em seguida cifrar este mesmo hash com uma chave privada. Este processo garante a autenticidade da mensagem, uma vez que foi cifrada com uma chave privada e ao decifrar a assinatura (Figura 5) com a respectiva chave pública, pode-se comparar o Hash decifrado com o Hash do texto plano original, se forem iguais, a integridade dos dados foi mantida.

Figura 4 – Assinatura digital



Fonte: Sutil et al. (2010)

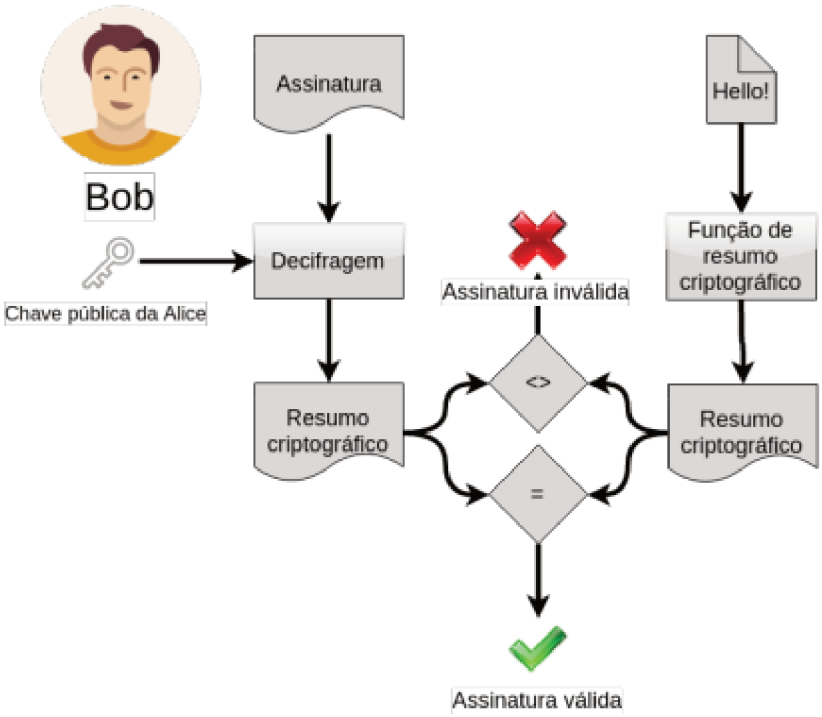
2.1.5 Certificado digital

Determinar quem possui a chave privada é um dos principais problemas na criptografia de chaves públicas. O uso de certificados digitais foi proposto para solucionar este problema. O certificado digital é um conjunto de informações que permitem identificar, de forma segura, o proprietário do par de chaves. Cada certificado contém a chave pública e a identificação da entidade que controla a respectiva chave privada.

Segundo (HOUSLEY; POLK, 2001), um certificado ideal deve conter uma série de características importantes:

- Ser um objeto puramente digital, que possa ser distribuido e processado automaticamente;
- Deve conter informações sobre o dono da chave privada;
- Deve ser fácil de determinar se o certificado foi recentemente emitido;

Figura 5 – Verificação de uma assinatura digital



Fonte: Trapp (2016)

- Deve ser criado por uma entidade confiável ao invés do próprio usuário que detém a chave privada;
- Uma vez que uma entidade confiável pode criar vários certificados, inclusive para um mesmo usuário, deve ser fácil diferenciá-los;
- Deve ser fácil determinar se o certificado foi forjado ou se é genuíno;
- Deve ser à prova de violação, de modo que ninguém consiga alterá-lo;
- Deve ser possível verificar de forma imediata se alguma informação no certificado não é mais válida;
- Deve-se poder determinar para quais aplicações o certificado é válido.

2.1.6 Cryptographic Message Syntax (CMS)

Na prática, uma assinatura digital não contém apenas o resumo criptográfico cifrado. Além do resumo criptográfico, podem ser incluídos também, atributos como data e local onde a assinatura foi realizada, certificado do signatário, sua cadeia de certificação e carimbo do tempo.

Uma assinatura digital pode ser representada pelo formado de assinatura CMS(Cryptographic Message Syntax), atualmente descrito pela RFC 5280, e trata-se de um formato binário, codificado em ASN.1 (SUTIL et al., 2010).

O formato binário ASN.1 é um padrão usado para descrever os dados que são transmitidos entre protocolos de telecomunicações, ou seja, não depende do software e hardware onde foi gerado ou será interpretado (HOUSLEY; POLK, 2001).

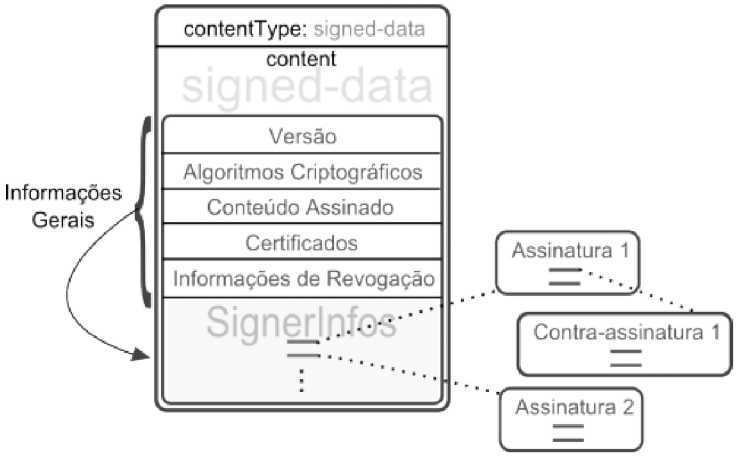
O CMS foi publicado em 1993 pelo RSA Data Security (RSA) como RSA Laboratories Technical Note, e é baseado na versão 1.5 do PKCS7, formato amplamente utilizado, de forma que existem muitas bibliotecas que permitem a geração e verificacão de arquivos PKCS7 (MOECKE, 2008).

O pacote CMS, ilustrado na figura 6 , mostra os atributos presentes em sua estrutura ASN.1. Para representar diferentes tipos de assinatura o CMS possui dois campos: contentType e content.

O contentType define o tipo de conteúdo, e o content é a estrutura do tipo especificado. No caso de uma assinatura digital, o contentType receberá o valor signed-data, que permite a representação de uma ou mais assinaturas digitais referentes ao mesmo documento através de outro campo chamado signerInfos (PEREIRA, 2012).

A figura 6 demonstra uma simplificação da estrutura CMS. Os campos agrupados pela chave *Informações Gerais*, incluiem dados relacionados a todas as assinaturas do pacote, como por exemplo o algoritmo de resumo criptográfico utilizado pelos signatários, e também o próprio conteúdo assinado. Podem ser incluídos nessa estrutura também, a cadeia de certificação do certificado do signatário, além das suas informações de revogação. O campo SignerInfos, contém informações referentes a apenas uma assinatura, ou seja, para cada assinatura realizada sobre um documento, existe um signerInfo correspondente no pacote CMS, estas assinaturas em paralelo são chamadas de co-assinaturas, que são assinaturas em paralelo. Entretanto, uma assinatura pode conter uma contra-assinatura, ou assinatura em série. Uma assinatura em série é feita sobre outra assinatura, e é utilizada como forma de reconhecimento de firma (PEREIRA, 2012).

Figura 6 – Estrutura CMS



Fonte: Pereira (2012)

2.2 Sandbox dos navegadores Web

De acordo com (REIS; BARTH; PIZANO, 2009) navegadores como Google Chrome e Mozilla Firefox funcionam em um modo conhecido como sandbox, um mecanismo de segurança que impede que chamadas nativas sejam feitas ao sistema operacional, tornando inviável o acesso direto a estes dispositivos bem como o acesso a chave privada para realizar assinaturas.

Construir um navegador Web seguro, é prioridade para a Google, (SYLVAIN, 2008), mas como se pode imaginar, um código perfeito é praticamente impossível de se obter em um projeto como o Google Chrome, por causa do seu tamanho e complexidade. Ainda segundo (SYLVAIN, 2008), um nagevador Web gasta a maior parte do tempo, lidando com a execução de código não permitido ou malicioso.

Praticamente, o Sandbox(Caixa de areia), é um mecanismo de segurança, utilizado para executar uma aplicação em um ambiente desconhecido. Se um atacante, consegue de alguma forma, se aproveitar de uma falha de segurança para executar um código malicioso na máquina, este modo impede que o dano seja causado no sistema, além de prevenir a modificação e leitura de arquivos do mesmo. Na grande maioria dos navegadores, toda a parte de renderização HTML e Javascript é isolada em um processo dentro deste módulo.

2.3 Tecnologias que possibilitam a assinatura através do navegador Web

O modo Sandbox dos navegadores, impossibilita que chamadas de sistema sejam feitas através dele por questões de segurança. Para realizar uma assinatura através de um navegador, é necessário ter acesso a chave privada do assinante, esta que pode estar disponível em um dispositivo criptográfico, como um Smartcard ou token. Porém, os navegadores não disponibilizam nenhuma interface de acesso a estes dispositivos.

Durante muito tempo, implementações fizeram uso da tecnologia *NPAPI* (SMEDBERG, 2015) para executar código nativo através do navegador, entretando, por questões de segurança, esta tecnologia foi descontinuada, obrigando desenvolvedores a encontrarem uma alternativa.

Na sequência do texto serão listadas algumas das alternativas existentes que solucionam este problema.

2.3.1 Applets Java

As applets em Java são uma maneira de incluir código nativo e executar nos clientes que visualizam uma página web. São programas pré-compilados feitos em Java, e a sua principal vantagem é que são mais independentes de navegador do que os scripts. Entretando os Applets em Java fazem com que o cliente dependa da JRE (Java Runtime Environment) instalada no seu computador, que é um pacote razoavelmente pesado (ORACLE, 2014).

O plugin NPAPI, tecnologia que possibilitava o uso dos Applets em Java nos navegadores Web foi descontinuado, tornando esta abordagem obsoleta.

2.3.2 ActiveX

ActiveX é um framework que já está se tornando obsoleto. Criado em 1996 pela Microsoft para a plataforma Windows. Controles ActiveX podem ser incluídos dentro de páginas web para realizar diversas ações, inclusive executar código nativo, similar aos Applets em Java. Porém os controles ActiveX, quando instalados, possuem privilégio para realizar qualquer ação que o usuário também possa realizar, como por exemplo, editar os registros do Windows, ou até mesmo apagar o disco rígido (Microsoft, 2016). Estas características tornam o ActiveX uma

tecnologia insegura. Além disso, os componentes ActiveX só são suportados pelo navegador Internet Explorer, e o Edge, navegador Web mais atual da Microsoft, não dará suporte a esta tecnologia (CHARLES; JACOB, 2015).

2.3.3 Websockets

Para reduzir o retardo nos serviços como o protocolo HTTP, surge o Websocket. WebSockets são um padrão ainda em desenvolvimento e evolução sob os cuidados da IETF. Uma API padrão para implementação deste protocolo está sendo formalizada pela W3C para que os navegadores ofereçam suporte ao mesmo (ESPOSITO, 2017).

Este protocolo tem como objetivo superar as limitações estruturais do protocolo HTTP, que é considerado ineficiente quando utilizado em aplicações que precisam permanecer conectadas com o servidor em uma conexão persistente. Por meio de um soquete TCP, este protocolo permite uma comunicação bidirecional entre os aplicativos e servidores Web.

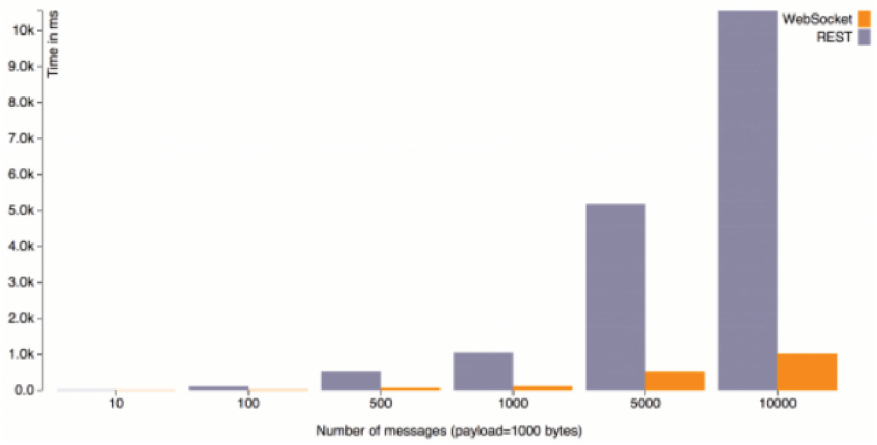
A principal vantagem dos WebSockets, se comparados ao protocolo HTTP, é que o WebSocket não precisa seguir o padrão request-response tradicional. Uma vez que o servidor abre uma conexão com um WebSocket, ambos os lados podem se comunicar assincronamente, e a conexão se mantém ativa até que o servidor a feche (ERKKILÄ, 2012).

A redução da redundância no protocolo WebSocket, implicou no menor consumo de recursos computacionais, bem como a latência que os usuários experimentam nas aplicações que o utilizam. Para enviar pequenas quantidades de arquivos grandes, as vantagens do Websocket não são muito atrativas, porém, quando se trata de muitos arquivos pequenos, o protocolo pode reduzir drasticamente o tempo de processamento dos dados (ERKKILÄ, 2012).

A figura 7, mostra uma comparação entre o protocolo HTTP e Websocket, os sistemas que utilizam WebSocket são capazes de processar mais mensagens em uma determinada faixa de tempo em comparação aos sistemas que são baseados no protocolo HTTP, desta forma aumentando a vazão de informação que o sistema é capaz de processar.

Um dos problemas do Websocket segundo (ERKKILÄ, 2012), é que além da falta dos headers do HTTP, o protocolo mascara suas comunicações para evitar alguns ataques de proxy, impedindo que ferramentas como antivírus e firewalls analisem o seu trafego, podendo causar falsos negativos a respeito da aplicação que utiliza esta comunicação.

Figura 7 – Comparação entre WebSocket e HTTP



Fonte: Gupta (2014)

2.3.4 Web Extensions e Native Messaging

Extensões são complementos adicionados ao navegador, e que podem modificar a funcionalidade do mesmo. Estes complementos são escritos utilizando as tecnologias padrões da Web, como Javascript, HTML e CSS. As extensões para o navegador Web Firefox, são construídas utilizando a API WebExtensions, esta API proporciona um sistema multi-navegador para desenvolvimento de extensões. Através desta API é possível criar uma extensão que seja compatível com o navegador Google Chrome e Opera, além disso, extensões escritas para estes navegadores, irão funcionar no Microsoft Edge com poucas mudanças (FTHIAGOGV; MOSTARD, 2017).

Através das extensões, é possível utilizar Native Messaging. Native Messaging é uma tecnologia que permite que a extensão troque mensagens com uma aplicação nativa instalada no computador de um usuário. Um exemplo comum são os gerenciadores de senha (Password managers), que utilizam Native Messaging para gerenciar os Stores(Gerenciadores de senha e certificados) e a criptografia das senhas. Além disso o Native Messaging permite que extensões acessem recursos de hardware que não estão disponsíveis através da API WebExtension, como por exemplo, dispositivos criptográficos (NULLDIVISOR, 2017).

2.3.5 Web Crypto API

Web Crypto API é uma tecnologia experimental que vem sendo desenvolvida pelos navegadores atuais, esta API apresenta uma interface de acesso a funções criptográficas primitivas

para criar sistemas que utilizam criptografia. Trata-se de uma abordagem nova, que permite até mesmo a realização de assinaturas e verificação da mesma. Entretanto, ainda que esta tecnologia possibilite estas operações, a Web Crypto API é incapaz de interagir com hardwares dedicados, como cartões inteligentes (smartcards), tokens USB ou geradores de aleatoriedade (GPRIMOLA HIKASHII, 2018).

2.4 Trabalhos relacionados

Além das tecnologias citadas na seção anterior, foram revisados alguns trabalhos que fizeram uso das mesmas para resolver problemas semelhantes. Como exemplo podemos citar o trabalho do autor (TRAPP, 2016), que na busca por uma alternativa aos applets para acesso a dispositivos, realizou uma implementação que utiliza um servidor HTTP local. O autor sugere como trabalhos futuros, o uso do protocolo de comunicação HTTPS, e também a implementação utilizando Websockets. Já o autor (SOARES, 2017), implementou um sistema de emissão de certificados online utilizando um servidor local HTTPS.

Estes dois trabalhos citam as dificuldades em relação a utilização de um servidor local, um deles é a necessidade de instalação do certificado da conexão SSL, que gera diversos problemas de segurança com os nagevadores web e programas antivírus. Ambos sugerem uma solução que não exija a instalação de um certificado.

3 Proposta e desenvolvimento do protótipo de assinaturas digitais através do navegador web

Por questões de segurança e desempenho, a arquitetura de plugins NPAPI que possibilitava o uso de Applets Java em navegadores Web, foi descontinuada. Os Applets em Java eram utilizados para realizar assinaturas digitais através do navegador, uma vez que podem executar código nativo e acessar os dispositivos criptográficos do computador do usuário. Com o fim do suporte à esta tecnologia pelos navegadores Web mais atuais, tornou-se necessária, uma busca por alternativas que tornassem possível a solução deste problema.

Na seção 2.3 foram listadas as tecnologias que possibilitam a realização de assinaturas digitais através de um navegador Web. Podemos citar os componentes ActiveX da Microsoft, que não são mais recomendados, uma vez que possuem acesso total a máquina do usuário e podem causar problemas de segurança, além disso funcionam apenas no Internet Explorer.

Além do ActiveX, existem também os WebSockets, através de um Websocket é possível instalar um servidor na máquina do usuário, que se comunica com o browser através da linguagem Javascript e um soquete TCP. Esta tecnologia é mais recomendada que o HTTP neste tipo de solução, uma vez que possibilita comunicação bidirecional, e é mais rápida, já que a redundância do protocolo HTTP, causada pelo tamanho do cabeçalho foi reduzida.

Como ponto negativo, podemos dizer que os WebSockets precisam estar executando como processos constantes na máquina do usuário, e se o processo for finalizado, não é possível fazer requisições para este servidor local. Além disso, a falta dos cabeçalhos HTTP e a forma como a comunicação é mascarada para evitar ataques de proxy (ERKKILÄ, 2012), faz com que ferramentas como anti-vírus e firewalls considerem o WebSocket como uma ferramenta maliciosa, muitas vezes impedindo o seu funcionamento.

É possível realizar conexões seguras a partir de um Websocket, porém, é necessário instalar um certificado no computador do usuário através de scripts, estes que também não são considerados seguros pelos sistemas operacionais, alguns navegadores inclusive, só permitem

que isso seja feito se uma configuração for ativada no seu painel de configurações (FISH, 2016).

Por fim, podemos citar as Extensões Web, ou Web Extensions, estes complementos podem ser adicionados no navegador do usuário, possibilitando a comunicação com um módulo nativo que realiza assinaturas. Através de um instalador, este módulo nativo deve ser registrado no sistema, e uma vez que foi registrado, possibilita que o navegador se comunique com o mesmo através da linguagem Javascript. Diferente dos WebSockets, um processo só é aberto quando uma chamada é feita, e removido somente quando a requisição é finalizada. As extensões Web baseadas na API WebExtensions, são compatíveis entre diversos navegadores, por exemplo, uma extensão desenvolvida para o navegador Google Chrome, pode ser utilizada nos navegadores Firefox e Opera praticamente sem que alterações precisem ser feitas.

Para este trabalho, a tecnologia Web Extensions foi escolhida por simplificar o desenvolvimento e não apresentar os problemas encontrados nas tecnologias como Applets, WebSockets e ActiveX. Extensões podem ser facilmente instaladas no computador do usuário, não apresentando problemas relacionados a antivírus e firewall por exemplo.

O formato CMS (Cryptographic Message Syntax) foi escolhido como modelo de assinatura por ser mais simples de desenvolver, porém, qualquer formato de assinatura existente poderia ter sido escolhido.

Portanto, o modelo de proposta deste trabalho consiste em uma solução que realize uma assinatura digital CMS através de uma extensão Web para o navegador Google Chrome no sistema operacional Windows.

3.1 Chrome Extensions

Nesta seção serão descritos os principais conceitos relacionados ao desenvolvimento de uma extensão. Embora o desenvolvimento de uma extensão seja muito parecido e compatível entre os diferentes browsers, iremos dar um foco maior no navegador Google Chrome, onde o protóripo desenvolvido irá funcionar.

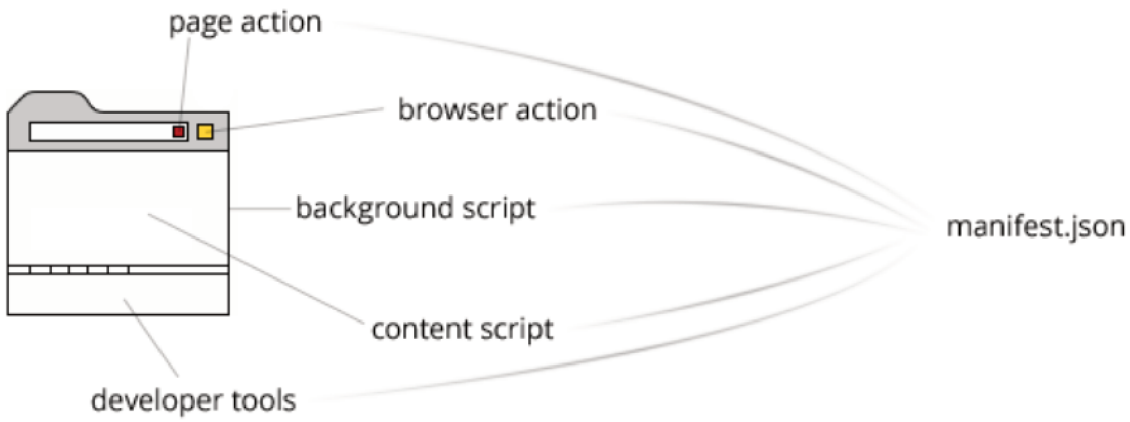
Na seção 2.3.4 vimos que extensões são complementos para um navegador web. Estes complementos podem ser desenvolvidos utilizando as tecnologias padrões da Web, como Javascript, HTML, CSS, e servem para customizar a experiência dos usuários com os navegadores (GOOGLE, 2018).

Uma extensão pode ser desenvolvida para diversos propósitos, como por exemplo alterar a cor de fundo de uma página ou até bloquear propagandas como o AdBlock, que é a extensão mais baixada nos navegadores da atualidade (ADBLOCK, 2018).

3.1.1 A arquitetura de uma extensão

Nesta seção veremos brevemente como é a arquitetura de uma extensão, quais arquivos devem estar presentes na sua estrutura e para que são utilizados. A figura 8 ilustra esta arquitetura.

Figura 8 – Arquitetura de uma extensão



Fonte: Google (2018)

3.1.1.1 manifest.json

O Arquivo manifest.json, é o único arquivo que obrigatoriamente deve estar presente em toda extensão. A figura 9 ilustra a estrutura de um arquivo manifest comum, nele estão informações muito importantes, como nome da extensão, versão, permissões utilizadas e ponteiros para outros arquivos que também serão incluídos e utilizados no pacote da aplicação (GOOGLE, 2018).

3.1.1.2 Content Scripts

Content scripts fazem parte da estrutura de uma extensão, e são incluídos no seu manifest. Escritos na linguagem Javascript, podem ser utilizados para interagir com as páginas Web. Este script é executado no contexto de uma página que foi carregada pelo navegador, como se fosse uma parte dela. Através do content script é possível ler detalhes das páginas

Figura 9 – Arquivo manifest.json

```
{
  "name": "My Extension",
  "version": "2.1",
  "description": "Gets information from Google.",
  "icons": { "128": "icon_128.png" },
  "background": {
    "persistent": false,
    "scripts": ["bg.js"]
  },
  "permissions": ["http://*.google.com/", "https://*.google.com/"],
  "browser_action": {
    "default_title": "",
    "default_icon": "icon_19.png",
    "default_popup": "popup.html"
  }
}
```

Fonte: Google (2018)

Web visitadas pelo navegador, escutar eventos, modificar o conteúdo ou layout, e até mesmo comunicar-se com outras extensões (GOOGLE, 2018).

Diferente dos scripts normais, o content script pode utilizar as APIs disponíveis apenas para uma WebExtension, realizar requisições XHR (Cross domain request) e trocar mensagens com o background script.

Neste trabalho o content script será utilizado para intermediar a comunicação entre a página que vai solicitar a assinatura e a extensão.

3.1.1.3 Background scripts

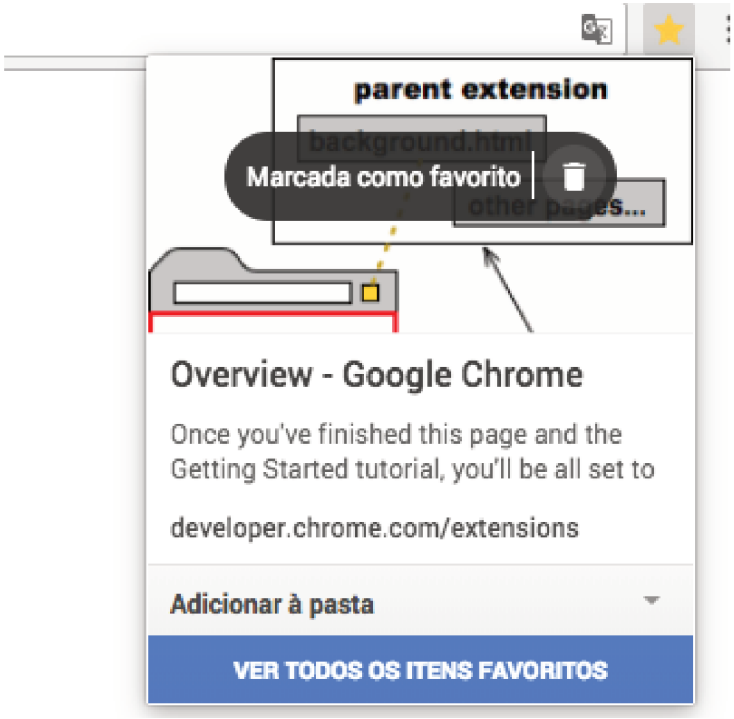
Background scripts também são escritos na linguagem Javascript, e podem ser utilizados para controlar o comportamento de uma extensão. Existem dois tipos de background scripts, os persistentes e os de evento (GOOGLE, 2018). A diferença entre eles é que os persistentes estão sempre executando e processando alguma informação, e os scripts de evento são inicializados para realizar uma ação específica e depois voltam para o seu estado inativo. A não ser que uma extensão utilize o background o tempo inteiro, é preferível que se use o background do tipo evento (GOOGLE, 2018).

Através do background script é possível realizar chamadas que nenhum outro script teria permissão, isto porque ele é executado em uma camada de baixo nível no navegador, tornando possível por exemplo, enviar mensagens nativas, funcionalidade essencial para o desenvolvimento deste trabalho.

3.1.1.4 A interface gráfica de uma extensão

Extensões podem criar uma representação gráfica através de páginas HTML, como pode ser visto na figura 10. Este botão que abre um popup é chamado de page action, através do page action, ao invés de um popup, é possível também, abrir uma nova aba com funcionalidades ou configurações da extensão, esta nova aba pode ter o seu próprio content script e background script (GOOGLE, 2018).

Figura 10 – Page action



Fonte: Autor

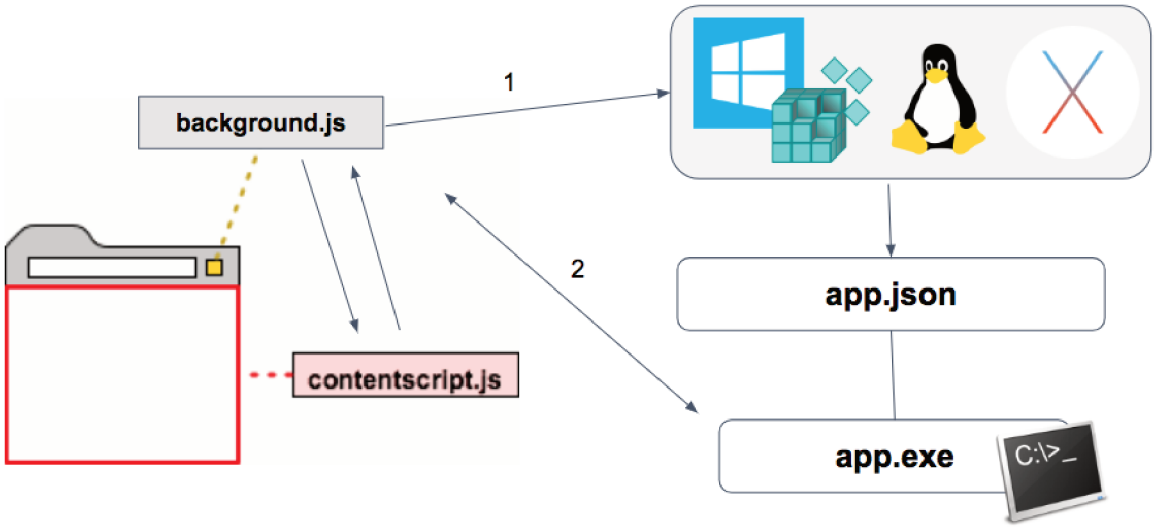
3.1.2 Native messaging

Na seção 2.3.4 vimos também, que através das extensões é possível utilizar uma tecnologia conhecida como Native Messaging. Esta tecnologia nos permite trocar mensagens com uma aplicação nativa instalada no computador de um usuário.

3.1.3 Realizando chamadas nativas através de uma extensão

Extensões podem trocar mensagens com aplicações nativas utilizando Native Messaging. A figura 11 demonstra como esta comunicação é realizada.

Figura 11 – Native messaging communication



Fonte: Autor

Primeiramente, a página HTML que está carregada no browser irá se comunicar com o content script da extensão, solicitando uma funcionalidade que utiliza Native Messaging, por exemplo, uma assinatura. Como apenas o background script é capaz de realizar chamadas nativas, esta solicitação será apenas repassada para ele.

O background script irá receber esta solicitação e tentar se conectar a aplicação nativa. Para que o background script possa realizar esta conexão, é necessário que o *host* esteja registrado no sistema. No caso do Windows, precisamos criar uma nova chave nos registros, e o valor desta chave deve apontar para o caminho de um arquivo manifest que contém todas as informações deste *host*.

O fato de registrar o *host* no sistema, faz com que ele se torne visível para o navegador. Desta forma, o background script irá conseguir se conectar à aplicação, isto porque o valor do registro deve apontar para um arquivo chamado host manifest (figura 12), e este arquivo contém o caminho para a aplicação instalada. Além de informar onde o *host* está localizado, este arquivo contém outras informações como o tipo de comunicação, nome da aplicação, e extensões que podem se comunicar através dele, ou seja, é possível limitar a comunicação a apenas uma

extensão (GOOGLE, 2018). Este processo será detalhado na criação do protótipo.

Figura 12 – Host manifest

```
{
  "name": "com.my_company.my_application",
  "description": "My Application",
  "path": "C:\\Program Files\\My Application\\chrome_native_messaging_host.exe",
  "type": "stdio",
  "allowed_origins": [
    "chrome-extension://knldjmfmopnpolahpmmgbagdohdnhkik/"
  ]
}
```

Fonte: Google (2018)

3.2 Protótipo funcional

Esta seção apresenta uma descrição de como foi desenvolvido o protótipo funcional deste trabalho, que será uma extensão para o navegador *Google Chrome*. Esta extensão será chamada de *Signer Web*.

3.2.1 Desenvolvimento

Nos tópicos a seguir, serão descritas as implementações realizadas. Para realizar uma assinatura, serão necessários os dados que serão assinados, e um certificado, portanto este protótipo deve ser capaz de listar os certificados instalados na máquina do usuário e assinar utilizando a chave privada associada ao certificado escolhido.

3.2.1.1 Content Script

Como pode ser visto na figura 13 foram incluídas duas funções no content script da extensão, estas funções escutam os eventos `sign` e `list_certificates`.

Quando a extensão receber um destes eventos, irá repassar a mensagem para o background script, através da função Javascript `chrome.runtime.sendMessage`, esta função é utilizada para realizar a comunicação entre a extensão e o background. A mensagem enviada é um JSON que contém as informações sobre a ação que será solicitada à extensão. Para o

Figura 13 – Content script

```
//Listener para o evento "list_certificates", irá repassar a mensagem para o background
document.addEventListener("list_certificates",  function(data)
{
    chrome.runtime.sendMessage('{"action":"list_certificates"}');
});

//Listener para o evento "sign", irá repassar a mensagem para o background
document.addEventListener("sign", function(event)
{
    var detail = event.detail;
    var json = JSON.parse(detail.input);
    json.action = "sign";
    json.certId = detail.idCertificado;
    chrome.runtime.sendMessage(JSON.stringify(json));
});
```

Fonte: Autor

evento `list_certificates` estamos apenas repassando o action `list_certificates`, já para o evento `sign` enviamos o action `sign`, os dados que serão assinados, e um identificador do certificado que irá realizar a assinatura.

3.2.1.2 Background Script

A função `chrome.runtime.sendMessage` envia uma única mensagem para as extensões que estão escutando, neste caso, o background script da extensão *Signer Web* está esperando esta mensagem através da função `chrome.runtime.onMessage`, reponsável por escutar as mensagens recebidas. A figura 14 demonstra como esta função foi implementada.

Na figura 14, podemos observar que a mensagem recebida é convertida em um objeto JSON, e caso o action recebido seja `list_certificates` ou `sign`, envia uma mensagem nativa para o *host* instalado no computador, através da função `chrome.runtime.sendNativeMessage`.

A função `chrome.runtime.sendNativeMessage`, é responsável por realizar a comunicação nativa com o *host*. Como primeiro parâmetro está sendo configurado o nome do *host*, que neste caso é signerweb. Este nome foi definido anteriormente no host manifest (figura 12), e como segundo parâmetro estamos repassando a mensagem que veio do content script. Por fim, o terceiro parâmetro é uma função de callback, que será chamada quando a requisição for completada.

Como podemos ver no código da figura 14, se por algum motivo a mensagem não

Figura 14 – Background script

```
chrome.runtime.onMessage.addListener(function(message, sender, sendResponse)
{
    var json = JSON.parse(message);

    if (sender.tab && sender.tab.id)
        json.tabid = sender.tab.id;
    else
       json.tabid = 0;

    switch(json.action) {
        case 'list_certificates':
        case 'sign':
            chrome.runtime.sendNativeMessage("signerweb", json, function (response) {
                if (response === undefined) {
                    onError("signer.not.installed", "O módulo nativo não foi instalado", json.tabid);
                    // O Aviso da badge deve ser global, por isso o tabId não é informado
                    chrome.browserAction.setBadgeText({text: "!"});
                    chrome.browserAction.setBadgeBackgroundColor({color: "#FAA300"});
                    response = {};
                }
                else {
                    chrome.browserAction.setBadgeText({text: ""});
                    onNativeMessage(response);
                }
            });
            break;
        default:
            onError("background.unknow.message", "A mensagem enviada para a extensão é desconhecida: \"" + json.action + "\".", json.tabid);
    }
});
```

Fonte: Autor

for enviada, o retorno desta função é *undefined*, e podemos dizer que a aplicação nativa não está instalada ou não pode ser encontrada, e caso contrário, sabemos que a mensagem foi bem sucedida e será retornada novamente para o content script, através da função `chrome.runtime.sendMessage`.

3.2.1.3 Função de callback

Sabemos que todo este processo deu início em uma página HTML qualquer que disparou um evento para a extensão solicitando uma ação. Este evento foi tratado pelo content script, e repassado para o background script, que por sua vez enviou a mensagem para a aplicação nativa (*host*) instalada na máquina do usuário. A mensagem foi processada pelo *host* e retornada para o background script, e o mesmo enviou a mensagem de volta para o content script.

Agora que a mensagem foi processada, o content script precisa responder a chamada desta página HTML, supondo que esta página queira exibir uma lista com os certificados instalados na máquina do cliente, o content script precisa enviar esta lista para a página. Neste trabalho isso foi feito através de uma função de callback.

Podemos observar na figura 15, que o nome da função de callback foi declarado no content script como uma variável constante com o valor `extension_module.callback`,

e este valor não pode ser alterado.

Quando a resposta do background é recebida pelo content script através da função `chrome.runtime.onMessage`, é verificado se a ação retornada é `sign` ou `list_certificates`, para que o retorno possa ser processado. Este retorno vem do background e é um objeto JSON que contém a resposta da solicitação anteriormente enviada, neste caso a lista de certificados instalados, ou uma assinatura.

Figura 15 – Função de callback

```
let callback_name = "extension_module.callback";
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
    var action = request.data.action;
    if(action == "list_certificates" ||  action == "sign") {
        /**
        * Chama o callback com os dados obtidos
        */
        code = callback_name + "(" + JSON.stringify(request.data) + ");";
        var script = document.createElement("script");
        script.textContent = code;
        (document.head||document.documentElement).appendChild(script);
        script.remove();
    }
```

Fonte: Autor

Definindo o nome da função de callback de forma constante, a página que desejar tratar o retorno do content script, precisa ter esta função declarada com o mesmo nome no seu códito Javascript, desta forma o content script irá chamar esta função enviando a solicitação processada como parâmetro. Isto pode ser feito porque o content script está lado a lado com o script da página, porém ele não tem como saber qual é a função de callback que será utilizada, por isso ela é definida de forma estática, e aqueles que quiserem receber o retorno, precisam declará-la com este nome.

A figura 16, exemplifica como esta função pode ser declarada na página HTML do cliente. O content script irá chamar esta função, informando o retorno como parâmetro `data`, através deste retorno é possível verificar qual foi o tipo da ação retornada, e escolher como tratá-lo. Neste caso para o retorno `sign`, será feito o download da assinatura, e para o retorno `list_certificate`, serão listados os certificados na página HTML.

Figura 16 – Função de callback cliente

```
//As chamadas caem nessa função de callback e são distribuidas de acordo com o action
extension_module.callback = function (data) {
    switch (data.action) {
        case "list_certificates":
            extension_module.updateCertificados(data.certificates);
            break;
        case "sign":
            extension_module.processarAssinaturas(data);
            break;
        case "error":
            extension_module.tratarError(data);
            break;
        default:
            alert("Mensagem recebida desconhecida.");
    }
}
```

Fonte: Autor

3.2.1.4 A aplicação nativa *host*

Até o presente momento foi descrito como é realizada a comunicação do ponto de vista da extensão, agora veremos como é feita a comunicação com o host instalado no computador do usuário.

Quando o background script envia uma mensagem nativa através da função `chrome.runtime.sendNativeMessage`, o Google Chrome irá buscar nos registros do Windows, pelo identificador incluído no primeiro parâmetro da função.

Uma chave deve ser criada no caminho `HKEY_LOCAL_MACHINE\SOFTWARE\Google\Chrome\NativeMessagingHosts\signerweb`, para que a extensão encontre o host manifest, descrito na figura 12. O arquivo host manifest, por sua vez, irá indicar a localização do *host*, que é o arquivo executável responsável por responder as solicitações da extensão.

Neste trabalho, o *host* foi desenvolvido na linguagem C++, por questões de afinidade com a linguagem e performance, porém poderia ter sido desenvolvido em outra linguagem, como Java ou Pyhton por exemplo. Todo o procedimento de registro e instalação da aplicação foi feito através de um instalador MSI. Este instalador é responsável por copiar os arquivos host e host manifest para um diretório específico, e criar uma chave nos registros que aponte para o arquivo host manifest.

Agora que o *host* já pode ser encontrado pela função *chrome.runtime.sendNativeMessage*, ele deve ser capaz de processar as solicitações enviadas pela extensão, e responde-las.

Como a comunicação da extensão com o *host* é feita através da entrada e saída de texto, a figura 17 demonstra como foi implementada a função *main*, responsável por receber os inputs e retornar as respostas à extensão.

Figura 17 – Função main

```
int main(int argc, char *argv[])
{
    RequestHandler handler;
    //Recebe a mensagem da extensão
    std::string request = handler.getRequestFromStream(stdin);

    std::string idExtension = "id_unknown";
    if (argc > 1)
        idExtension = argv[1];
    //Processa a requisição
    std::string response = handler.processRequest(request, idExtension);

    //Devolve a mensagem à extensão.
    handler.processOutput(response, std::cout);
}
```

Fonte: Autor

Os métodos `getRequestFromStream` e `processOutput`, são responsáveis respectivamente por ler os dados de entrada, e retornar os dados de saída.

O método `processRequest` é o core da aplicação, porque ele é o responsável por listar os certificados e realizar uma assinatura, os dados recebidos como entrada são passados para ele como parâmetro. Este método irá verificar o tipo de ação solicitada e chamar a função responsável para responder esta ação (figura 18).

Figura 18 – Função processRequest

```
if (action == "list_certificates") {
    response = Signer::criarJsonListaCertificados(tabid);
}
else if (action == "sign") {
    response = Signer::assinar(root["data"].asString(), root["certId"].asString(), tabid);
}
else {
    throw std::exception("Ação desconhecida");
}
```

Fonte: Autor

Caso a solicitação recebida seja `list_certificates`, a classe Signer irá construir uma mensagem no formato JSON com a lista dos certificados instalados. Porém, se a

solicitação recebida for `sign`, a classe irá realizar uma assinatura sobre os dados informados, utilizando a chave privada associada ao certificado escolhido. A resposta desta solicitação será enviada para a extensão.

Este é o ponto principal deste trabalho, por ser uma aplicação nativa instalada no computador do usuário, o *host* tem acesso aos dispositivos criptográficos e consequentemente, às chaves privadas gravadas dentro deles. A classe Signer utilizada para realizar a assinatura, gerar a lista de certificado, e acessar os dispositivos criptográficos, pertence a uma biblioteca privada fornecida pela empresa BRy Tecnologia. Esta biblioteca foi escolhida por questões de afinidade e viabilidade de integração com a linguagem C++.

3.2.1.5 O Protótipo

Agora que todas as implementações foram realizadas, veremos como este protótipo pode ser utilizado na prática. Primeiramente é necessário instalar a extensão desenvolvida, isto pode ser feito de duas formas, registrando a extensão na loja de extensões do navegador Google Chrome, ou instalando no modo desenvolvedor. Como a primeira opção é paga, a segunda opção foi escolhida para dar continuidade a este trabalho.

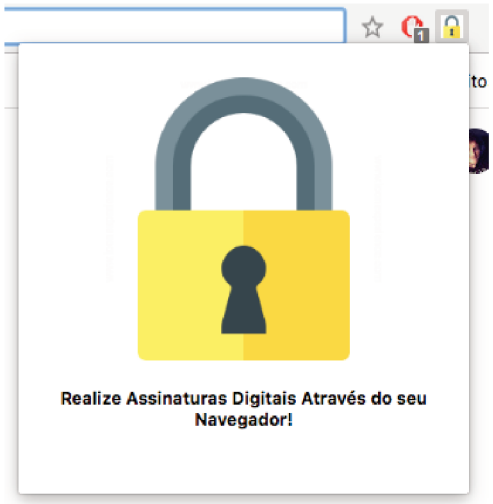
Para instalar a extensão no modo de Desenvolvedor basta acessar as configurações do Google chrome e navegar pelas opções `Configuracoes->Mais-ferramentas->Extensoes`, habilitar a opção *Modo desenvolvedor*, clicar no botão *Carregar sem compactação* e selecionar o diretório onde se encontra o arquivo manifest da extensão, desta forma a extensão será instalada (19).

Agora que a extensão está instalada, podemos criar uma página HTML para utilizar suas funcionalidades. Observa-se na figura 20 que foi criada uma página HTML com uma lista de certificados, e um botão ao lado para atualizar esta lista, além disso foi incluído um botão para gerar uma assinatura.

Quando clicamos neste botão que atualiza a lista de certificados, o mesmo irá disparar um evento com o identificador `list_certificates`. A extensão irá tratar este evento, realizar a comunicação nativa com o *host*, e responder através do função de callback, a função de callback irá preencher a lista de certificado com os nomes dos certificados retornados na mensagem da extensão, e também irá guardar o identificador de cada certificado.

Quando o botão Gerar assinatura é clicado, este irá disparar um evento com o

Figura 19 – Extensão instalada



Fonte: Autor

Figura 20 – Página HTML do protótipo

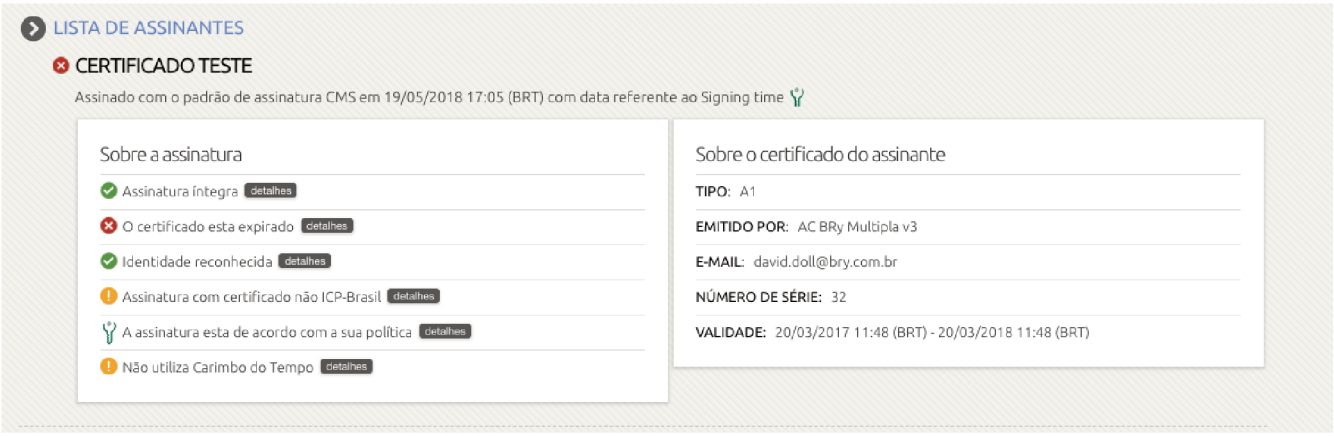
Fonte: Autor

identificador `sign`, porém, como parâmetro deste evento, irá enviar também o identificador do certificado selecionado, e o dado que deseja assinar, que neste trabalho será utilizado um dado estático, já definido no código. Assim como o evento `list_certificates`, a resposta será tratada na função de callback, mas neste caso, será feito o download da assinatura.

3.2.1.6 Verificação da assinatura gerada

Foi utilizado um verificador de assinaturas online para validar a assinatura gerada, e se o certificado selecionado foi utilizado. Através da figura 21 podemos verificar que a assinatura está íntegra, autêntica, e que foi assinada pelo certificado utilizado no protótipo. O único problema que aparece na assinatura, é que o certificado utilizado está expirado, porém o certificado de fato estava expirado.

Figura 21 – Verificação da assinatura gerada



Fonte: Autor

3.2.1.7 Objetivos e segurança

Através do protótipo desenvolvido, podemos perceber como fica fácil para uma aplicação Web qualquer utilizar esta extensão, basta solicitar que os seus usuários instalem a extensão e a aplicação nativa, e já será possível comuicar-se para gerar assinaturas.

Um dos objetivos específicos deste trabalho estava relacionado com a permissão de administrador para instalar a aplicação. Uma das maiores queixas dos usuários de WebSockets e servidores locais, é a necessidade desta permissão para realizar a instalação. Isto não é necessário para a instalação desta extensão, já que podemos instalar a aplicação nativa e adicionar o registro, na pasta local do sistema, que não solicita este tipo de permissão.

Outra questão importante, é que o processo de comunicação entre a extensão e o *host*, só é executado quando uma requisição é realizada, e após a finalização o processo é deletado. Isto gera uma grande diferença de usabilidade, já que servidores precisam estar sempre rodando. Se um usuário instalou um servidor, ou WebSocket, mas por algum motivo o serviço não está

sendo executado, a aplicação irá sempre acusar que a instalação não foi feita, solicitando que o usuário reinstale diversas vezes.

Por ser uma comunicação entre processos, também evitamos problemas com anti-vírus. Um dos problemas relacionados ao WebSocket e servidores locais, é que muitas vezes a conexão não é segura, e quando é, o anti-vírus suspeita do que está sendo enviado e recebido, por este motivo muitas vezes impede a aplicação de funcionar.

Diferente dos servidores locais e Websockets, a comunicação da extensão com a aplicação nativa é feita através de processos locais que executam no computador do usuário, e não serviços. Apesar de não utilizar protocolos seguros como o SSL, podemos dizer que esta comunicação é segura, e que apenas um programa malicioso com privilégios de administrador conseguiria interceptar a troca de mensagens entre a extensão e a aplicação nativa. Portanto, a não ser que o usuário instale algum software malicioso no seu computador, não teria como tornar esta comunicação insegura. Além disso, foi definido no host manifest, que apenas a extensão Signer Web pode trocar mensagens com o *host*, sendo assim, não existe a possibilidade de outra extensão tentar se conectar a ele.

É importante informar que toda página pode solicitar assinaturas, ou a lista de certificados instalados. Como o conent script é injetado em qualquer página do navegador, qualquer site poderia tentar realizar uma assinatura com o certificado do usuário, no entanto, podemos solucionar este problema permitindo apenas que domínios específicos façam este tipo de requisição. Além disso, o content script pode ser ofuscado, de forma que atacantes não poderão saber qual é o nome do evento que deve ser enviado.

4 Conclusão

Este trabalho teve como problema inicial a descontinuação da tecnologia Java Applet que possibilitava o acesso a dispositivos criptográficos através de um browser. O fim desta tecnologia criou a necessidade de busca por uma nova forma de realizar assinatura através dos navegadores Web. As tecnologias existentes que possibilitam a solução deste problema são: o Java Applet, componentes ActiveX, WebSockets e WebExtensions.

Por questões de usabilidade, segurança e facilidade de desenvolvimento, a tecnologia WebExtension foi escolhida, e um protótipo que atende aos objetivos específicos deste trabalho foi desenvolvido.

Podemos confirmar que extensões podem ser utilizadas para realizar assinaturas e acessar dispositivos criptográficos através de um navegador web, utilizando a tecnologia Native Messaging dos navegadores. Estas extensões podem ser facilmente integradas a qualquer aplicação web, de forma segura.

Através do desenvolvimento deste protótipo, capaz de realizar assinaturas CMS e listar os certificados instalados no computador do usuário ou dispositivos criptográficos, podemos concluir que as extensões são uma ótima forma de resolver este problema. Além de vantagens de usabilidade, podemos citar diversas outras, como por exempo, a segurança, solução de problemas relacionados a programas anti-vírus e a possibilidade de instalação sem permissão de administrador, que eram os principais objetivos deste trabalho.

4.1 Trabalhos futuros

A extensão que soluciona o problema apresentado por este trabalho, foi implementada com foco no sistema operacional Windows, e para o navegador Google Chrome. A API de desenvolvimento de extensões WebExtensions, possibilita que uma mesma extensão possa ser compatível com diversos navegadores, como o Firefox, Edge e opera. Bastaria que alguns pequenos ajustes fossem realizados.

Como tema para trabalhos futuros, seria interessante o desenvolvimento de uma extensão que fosse compatível com estes navegadores e outros sistemas operacionais. Como

o módulo nativo foi escrito em C++, o código poderia ser aproveitado, tornando necessário apenas alguns ajustes em relação a forma comos os certificados são acessados em outros sistemas operacionais.

Referências

ADBLOCK. *https://getadblock.com/*. 2018. Disponível em: <hhttps://developer.chrome.com-/extensions>. Acesso em: 20 maio 2018. Citado na página 33.

ADOBE. *Adobe Flash Player*. 2017. Disponível em: <http://get.adobe.com/br/flashplayer-/about/>. Acesso em: 17 out. 2017. Citado na página 16.

ALFRED, J. M.; PAUL, C. v. O.; SCOTT, A. V. Handbook of applied cryptography. *Massachusetts Institute of Technology*, p. 560, 1996. Citado na página 21.

CHARLES, M.; JACOB, R. *A break from the past, part 2: Saying goodbye to ActiveX, VBScript, attachEvent...* 2015. Disponível em: <https://blogs.windows.com/msedgedev/2015/05/06/a-break-from-the-past-part-2-saying-goodbye-to-activex-vbscript-attachevent/>. Acesso em: 24 out. 2017. Citado na página 28.

ELTIGER. *Criptografia, assinatura digital e alguns outros conceitos*. 2008. Disponível em: <https://eltiger.wordpress.com/2008/10/12/ciptografia-assinatura-digital-e-alguns-outros-conceitos/>. Acesso em: 12 out. 2008. Citado na página 22.

ERKKILÄ, J.-P. Websocket security analysis. *Aalto University School of Science*, p. 2–3, 2012. Citado 2 vezes nas páginas 28 e 31.

ESPOSITO, D. *Cutting Edge: Compreendendo o poder dos WebSockets*. 2017. Disponível em: <https://msdn.microsoft.com/pt-br/magazine/hh975342.aspx>. Acesso em: 18 out. 2017. Citado na página 28.

FERREIRA, P. R. G. A assinatura digital é assinatura formal. *Disponıvel em< http://www. cbeji. com. br>. Acesso em 17 out. 2017*, v. 20, n. 10, 2001. Citado na página 15.

FISH, B. *Access localhost via Microsoft Edge extension background page*. 2016. Disponível em: <https://developer.microsoft.com/en-us/microsoft-edge/platform/issues/8984919/>. Acesso em: 25 out. 2017. Citado na página 32.

FLR, h. *Websockets*. 2017. Disponível em: <https://developer.mozilla.org/pt-BR/docs-/WebSockets>. Acesso em: 18 out. 2017. Citado na página 17.

FTHIAGOGV; MOSTARD. *O que são extensões?* 2017. Disponível em: <https://developer-.mozilla.org/pt-BR/Add-ons/WebExtensions/What_are_WebExtensions>. Acesso em: 25 out. 2017. Citado na página 29.

GANDINI, J. A. D.; SALOMÃO, D. P. d. S.; JACOB, C. A segurança dos documentos digitais. *Disponıvel em< http://www. jus. com. br>. Acesso em: 17 out. 2017*, v. 11, 2001. Citado na página 15.

GOOGLE. *Getting Started Tutorial*. 2018. Disponível em: <hhttps://developer.chrome-.com/extensions>. Acesso em: 20 maio 2018. Citado 5 vezes nas páginas 32, 33, 34, 35 e 37.

GPRIMOLA HIKASHII, c. *Web Crypto API*. 2018. Disponível em: <https://developer.mozilla-.org/pt-BR/docs/Web/API/Web_Crypto_API>. Acesso em: 28 jun. 2018. Citado na página 30.

GUPTA, A. *REST vs WebSocket Comparison and Benchmarks*. 2014. Disponível em: <https://planet.jboss.org/post/REST_vs_WebSocket_comparison_and_benchmarks>. Acesso em: 24 out. 2017. Citado na página 29.

HOUSLEY, R.; POLK, T. *Planning for PKI: best practices guide for deploying public key infrastructure*. [S.l.]: John Wiley & Sons, Inc., 2001. Citado 2 vezes nas páginas 23 e 25.

MENKE, F. Assinaturas digitais, certificados digitais, infra-estrutura de chaves públicas brasileira e a icp alemã. *Revista de Direito do Consumidor*, v. 12, n. 48, 2003. Citado na página 15.

Microsoft. *Controles ActiveX desatualizados*. 2016. Disponível em: <https://docs.oracle.com-/javase/tutorial/deployment/applet/getStarted.html>. Acesso em: 24 out. 2017. Citado na página 27.

MOECKE, C. T. Assinatura digital de documentos eletronicos na icp-brasil. Universidade Federal de Santa Catarina, 2008. Citado na página 25.

NULLDIVISOR. *Native messaging*. 2017. Disponível em: <https://developer.mozilla.org/pt-BR/Add-ons/WebExtensions/What_are_WebExtensions>. Acesso em: 25 out. 2017. Citado na página 29.

ORACLE. *Getting Started With Applets*. 2014. Disponível em: <https://docs.oracle.com/javase-/tutorial/deployment/applet/getStarted.html>. Acesso em: 24 out. 2017. Citado na página 27.

Oracle. *Oracle Finally Kills Java Browser Plugin*. 2016. Disponível em: <https://www-.trendmicro.com/vinfo/us/security/news/vulnerabilities-and-exploits/oracle-finally-kills-java-browser-plugin>. Acesso em: 17 out. 2017. Citado na página 16.

PEREIRA, A. S. d. S. Aplicabilidade do padrão de assinatura pades no Âmbito da icp-brasil. Universidade Federal de Santa Catarina, 2012. Citado 3 vezes nas páginas 19, 25 e 26.

REIS, C.; BARTH, A.; PIZANO, C. Browser security: lessons from google chrome. *Queue*, ACM, v. 7, n. 5, p. 3, 2009. Citado 2 vezes nas páginas 16 e 26.

SCHNEIER, B. *Applied cryptography: protocols, algorithms, and source code in C*. [S.l.]: john wiley & sons, 2007. Citado na página 19.

SKYPE. *O que é o Skype Web Plugin e como faço para instalá-lo?* 2017. Disponível em: <https://support.skype.com/pt-br/faq/FA12316/o-que-e-o-skype-web-plugin-e-como-faco-para-instala-lo>. Acesso em: 17 out. 2017. Citado na página 16.

SMEDBERG, B. *NPAPI Plugins in Firefox*. 2015. Disponível em: <https://blog.mozilla.org-/futurereleases/2015/10/08/npapi-plugins-in-firefox/>. Acesso em: 17 out. 2017. Citado 2 vezes nas páginas 16 e 27.

SMITH, D. *JDK 9 e o Plug-in Java*. 2017. Disponível em: <https://www.java.com-/pt_BR/download/faq/jdk9_plugin.xml>. Acesso em: 17 out. 2017. Citado na página 16.

SOARES, P. Alternativa para emissÃo de certificados por autoridade certificadora online. *Disponıvel em<https://tcc.inf.ufsc.br/public/projetos.xhtm>. Acesso em 21 out. 2017*, 2017. Citado na página 30.

STALLINGS, W. *Criptografia e segurança de redes: princípios e práticas*. [S.l.]: Pearson Prentice Hall, 2008. Citado 4 vezes nas páginas 15, 19, 20 e 22.

SUTIL, J. M. et al. *Introdução a Infraestrutura de Chaves Públicas e Aplicações (ICPEDU)*. [S.l.]: Escola Superior de Redes RNP, 2010. Citado 5 vezes nas páginas 20, 21, 22, 23 e 25.

SYLVAIN, N. *A new approach to browser security: the Google Chrome Sandbox*. 2008. Disponível em: <https://blog.chromium.org/2008/10/new-approach-to-browser-security-google-.html>. Acesso em: 22 out. 2017. Citado na página 26.

Sylvain, Nicolas. *The Final Countdown for NPAPI*. 2014. Disponível em: <https://blog-.chromium.org/2014/11/the-final-countdown-for-npapi.html>. Acesso em: 17 out. 2017. Citado na página 16.

TRAPP, M. Uma alternativa aos applets em java para acesso a dispositivos. *Disponıvel em<https://tcc.inf.ufsc.br/public/projetos.xhtm>. Acesso em 21 out. 2017*, v. 20, n. 10, 2016. Citado 2 vezes nas páginas 24 e 30.

Anexos

ANEXO A – Código fonte da extensão

```
1  //Nome da funcao de callback
2  let callback_name = "extension_module.callback";
3
4  //Codigo que sera injetado na pagina sempre que for recarregada
5  let code = `
6  var signer_web = {};
7
8  signer_web.listarCertificados = function()
9  {
10     var event = document.createEvent("Event");
11     event.initEvent('list_certificates', true, true);
12     document.dispatchEvent(event);
13 }
14
15 signer_web.assinar = function(certificado, inputData)
16 {
17     var event = new CustomEvent("sign", {detail: {
18         idCertificado: certificado,
19         input: inputData
20     }});
21     event.initEvent('sign', true, true);
22     document.dispatchEvent(event);
23 }
24 `
25
26 //Este bloco e responsavel por injetar o script acima
27 var s = document.createElement('script');
28 s.text = code;
29 (document.head||document.documentElement).appendChild(s);
30 s.parentNode.removeChild(s)
31
32
33 //Listener para o evento "list_certificates", ira repassar a mensagem para
```

```
    o background
document.addEventListener("list_certificates",  function(data)
{
    chrome.runtime.sendMessage('{"action":"list_certificates"}');
});

//Listener para o evento "sign", ira repassar a mensagem para o background
document.addEventListener("sign", function(event)
{
    var detail = event.detail;
  var json = JSON.parse(detail.input);
  json.action = "sign";
  json.certId = detail.idCertificado;
  chrome.runtime.sendMessage(JSON.stringify(json));
});


chrome.runtime.onMessage.addListener(function(request, sender, sendResponse
    ) {
    var action = request.data.action;
    if(action == "list_certificates" ||  action == "sign") {
        /**
        * Chama o callback com os dados obtidos
        */
        code = callback_name + "(" + JSON.stringify(request.data) + ");";
        var script = document.createElement("script");
        script.textContent = code;
        (document.head||document.documentElement).appendChild(script);
        script.remove();
    }
    else if( action == "error" ) {
        var data = {};
        data.action = "error";
        data.descricao = request.data.descricao;
        data.chave = request.data.chave;

        /*
        * Chama o callback com a mensagem recem criada
        */
```

```
71          code = callback_name + "(" + JSON.stringify(data) + ");";
72          var script = document.createElement("script");
73          script.textContent = code;
74          (document.head||document.documentElement).appendChild(script);
75          script.remove();
76      }
77      else {
78          /*
79          * Cria uma mensagem de erro.
80          */
81          var data = {};
82          data.action = "error";
83          data.descricao = "A mensagem recebida pela extensao nao possui um
                action conhecido: \"" + action + "\".";
84          data.chave = "content_script.unknow.message";
85
86          /*
87          * Chama o callback com a mensagem recem criada
88          */
89          code = callback_name + "(" + JSON.stringify(data) + ");";
90          var script = document.createElement("script");
91          script.textContent = code;
92          (document.head||document.documentElement).appendChild(script);
93          script.remove();
94      }
95  });
```

Listing A.1 – contentscript.js

```
1  function onNativeMessage(message) {
2    try {
3      if (message.tabid) {
4        chrome.tabs.sendMessage(message.tabid, {data: message}, function(
             response) {});
5      } else if (message.action == "error") {
6        chrome.tabs.query({active: true, currentWindow: true}, function(tabs)
             {
7          if (tabs.length != 0) {
8            chrome.tabs.sendMessage(tabs[0].id, {data: message}, function(
                 response) {});
```

```
9         }
10      });
11    }
12  }
13  catch (ex)
14  {
15    var errorMessage = {};
16    errorMessage.action = "error";
17    errorMessage.descricao = "Excecao ao fazer chamada nativa."
18    errorMessage.chave = "signer.native.exception";
19    errorMessage.ex = ex;
20    // So sera exibida se a pagina estiver em foco enquanto o usuario
         utiliza.
21    chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
22      if (tabs.length != 0) {
23        chrome.tabs.sendMessage(tabs[0].id, {data: errorMessage}, function(
             response) {});
24      }
25    });
26  }
27 }
28
29 function onError(key, description, tabid) {
30   var message = {};
31   message.action = "error";
32   message.descricao = description;
33   message.chave = key;
34   chrome.tabs.sendMessage(tabid, {data:message}, function(response) {});
35 }
36
37 chrome.runtime.onMessage.addListener(function(message, sender, sendResponse
     )
38 {
39    var json = JSON.parse(message);
40
41    if (sender.tab && sender.tab.id)
42        json.tabid = sender.tab.id;
43    else
44       json.tabid = 0;
```

```
45
46    switch(json.action) {
47      case 'list_certificates':
48      case 'sign':
49        chrome.runtime.sendNativeMessage("signerweb", json, function (
              response) {
50          if (response === undefined) {
51            onError("signer.not.installed", "O modulo nativo nao foi
                  instalado", json.tabid);
52            // O Aviso da badge deve ser global, por isso o tabId nao e
                  informado
53            chrome.browserAction.setBadgeText({text: "!"});
54            chrome.browserAction.setBadgeBackgroundColor({color: "#FAA300"});
55            response = {};
56          }
57          else {
58            chrome.browserAction.setBadgeText({text: ""});
59            onNativeMessage(response);
60          }
61        });
62        break;
63      default:
64        onError("background.unknow.message", "A mensagem enviada para a
              extensao e desconhecida: \"" + json.action + "\".", json.tabid);
65    }
66  });
```

Listing A.2 – backgrond.js

```
1  {
2    "key" : "
       MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAlQJAuUwPdHqBKKoaa3sarntCyGRYkB0kiKuk
       /a7rnuPZDWzU+aY5fgVAH4E7GpP3Ol3HQ6cnHtq5MNPxRZTQtYAMJ+jEa1xwIL+
       A3fId4K3icQOkkBvOnRUPzNBghuHSGGn+lSYfCgkTN7/25+
       X0xepnAwkDMuvXv0PAXivg1sqVlCUGaK68NbuULThHBTUss+W0QmgoZNWCtiE1wl4L2a6l
       +SDtxFNxug40ZBEIfJBv8fwMl1SCcXQvFyTxh+1
       dgzvALXynIapjbVHOBw8RSFbxXRDCYupP0v+CLm/mvb2wTHifwIDAQAB",
3    "name": "Signer Web",
4    "version": "1.0.0",
5    "version_name": "1.0.0",
```

```
6  "manifest_version": 2,
7  "content_security_policy": "script-src 'self' 'unsafe-eval'; object-src '
       self'",
8  "description": "Trabalho de conclusao de curso. David Grechi Doll",
9  "browser_action": {
10   "default_icon": {
11       "19": "assets/marca-signer-150dpi-grayscale.png",
12       "38": "assets/marca-signer-150dpi-grayscale.png"
13   },
14   "default_title": "Signer Web",
15   "default_popup": "main.html"
16 },
17 "devtools_page": "devtools.html",
18
19 "background": {
20   "scripts": ["background.js"],
21   "persistent": false
22 },
23 "icons": {
24   "128": "assets/marca-signer-150dpi.png"
25 },
26 "permissions": [
27                   "nativeMessaging",
28                   "tabs",
29                   "activeTab",
30                   "background",
31                   "storage",
32                   "<all_urls>"
33               ],
34 "content_scripts": [{
35       "matches": ["<all_urls>"],
36       "js": ["assets/jquery-3.1.1.min.js","content_script.js"],
37       "run_at": "document_start"
38   }],
39
40 "web_accessible_resources": [
41   "assets/permission.jpg",
42   "assets/download.jpg",
43   "assets/marca-signer-150dpi.png",
```

```
44      "assets/instalador.png"
45    ]
46  }
```

Listing A.3 – manifest.json

ANEXO B – Código fonte da aplicação nativa

```
#include "..\core\RequestHandler.h"
#include <iostream>
#include "..\core\Signer.h"

int main(int argc, char *argv[])
{
  RequestHandler handler;
  //Recebe a mensagem da extensao
  std::string request = handler.getRequestFromStream(stdin);

  std::string idExtension = "id_unknown";
  if (argc > 1)
    idExtension = argv[1];
  //Processa a requisicao
  std::string response = handler.processRequest(request, idExtension);

  //Devolve a mensagem a extensao.
  handler.processOutput(response, std::cout);
}
```

Listing B.1 – main.cpp

```
#pragma once
#include "RequestHandler.h"
#include "Signer.h"
#include <fcntl.h>
#include <io.h>

RequestHandler::RequestHandler() {
  _setmode(_fileno(stdin), _O_BINARY);
  _setmode(_fileno(stdout), O_BINARY);
}

//Funcao que le o JSON que vem da extensao.
```

```
13 std::string RequestHandler::getRequestFromStream(FILE* fp)
14 {
15   unsigned int requestlength = 0;
16   for (unsigned int i = 0; i < 4; ++i)
17   {
18     unsigned int read_char = getc(fp);
19     requestlength = requestlength | (read_char << i * 8);
20   }
21
22   std::string request;
23   request.reserve(requestlength);
24   for (unsigned int i = 0; i < requestlength; ++i)
25   {
26     request.insert(request.begin() + i, getc(fp));
27   }
28
29   return request;
30 }
31
32 //Funcao responsavel por verificar a requisiao feita e construir o JSON de
     resposta.
33 std::string RequestHandler::processRequest(const std::string& request,
     const std::string& idExtension)
34 {
35   std::string response;
36   std::string action;
37   std::string tabid;
38
39   try
40   {
41     Json::Reader reader;
42     Json::Value root;
43     bool ok = reader.parse(request, root);
44
45     if (!ok)
46       throw std::exception("Falha ao fazer parse do JSON");
47
48     action = root["action"].asString();
49     tabid = root["tabid"].asString();
```

```
50
51      if (action == "list_certificates") {
52        response = Signer::criarJsonListaCertificados(tabid);
53      }
54      else if (action == "sign") {
55        response = Signer::assinar(root["data"].asString(), root["certId"].
              asString(), tabid);
56      }
57      else {
58        throw std::exception("Acao desconhecida");
59      }
60    }
61    catch (std::exception& ex)
62    {
63      response = "{" +
64        Signer::createJsonPair("action", "error", true) + "," +
65        Signer::createJsonPair("descricao", ex.what(), true) + "," +
66        Signer::createJsonPair("chave", "error,", true) +
67        Signer::createJsonPair("codigo", "-1," , true)+
68        Signer::createJsonPair("tipoRequisicao", action, true) +
69      "}";
70    }
71    return response;
72  }
73
74  //Funcao responsavel por devolver o JSON a extensao.
75  void RequestHandler::processOutput(const std::string& response, std::
        ostream& output)
76  {
77    // Collect the length of the message
78    unsigned int len = response.length();
79
80    // Now we can output our message
81    output << char(len >> 0)
82      << char(len >> 8)
83      << char(len >> 16)
84      << char(len >> 24);
85
86    output << response;
```

```
87 }
```

Listing B.2 – RequestHandler.cpp

```
1  #include "Signer.h"
2  #include <Bry\X509\StoreFactory.h>
3  #include "Bry\X509\CertificateChainFactory.h"
4  #include <Bry\Base\UtilConversor.h>
5  #include <vector>
6  #include <algorithm>
7
8  std::vector <Bry::X509::CertificateIcpBrasilSmartPtr> listarCertificados()
9  {
10   //Funcao que lista os certificados do windows e leitoras/tokens.
11 }
12
13 //Funcao que retorna o JSON de reposta para a requisicao "list_certificates
       "
14 std::string Signer::criarJsonListaCertificados(const std::string &tabid)
15 {
16   std::vector <Certificates> certificados = listarCertificados();
17   std::string listaCertificados;
18   try
19   {
20     std::string tabID = tabid == "" ? "-1" : tabid;
21
22     listaCertificados = "{" +
23       createJsonPair("action", "list_certificates", true) + "," +
24       createJsonPair("tabid", tabID, false) + "," +
25       "\"certificates\":[";
26
27     for (unsigned int index = 0; index < certificados.size(); ++index)
28     {
29       listaCertificados += "{" +
30         createJsonPair("name", certificados.at(index)->getSubject().
              CommonName, true) + "," +
31         createJsonPair("certId", certificados.at(index)->getCertId(), true)
              + "}";
32
33       if (index < certificados.size() - 1)
```

```
34          listaCertificados += ",";
35        }
36        listaCertificados += "]}";
37      }
38      catch (std::exception& ex)
39      {
40        throw ex;
41      }
42      return listaCertificados;
43  }
44
45  std::string assinarInternal(const std::string& data, const std::string&
        idCertificado)
46  {
47    //Funcao que assina e retorna o pacote CMS no formato base64.
48  }
49
50  //Funcao que retorna o JSON de resposta para a requisicao "sign".
51  std::string Signer::assinar(std::string data, std::string certId, std::
        string tabid)
52  {
53    std::string message;
54    std::string tabID = tabid == "" ? "-1" : tabid;
55    std::string assinatura = assinarInternal(data, certId);
56    try
57    {
58      message = "{" +
59        createJsonPair("action", "sign",true) + ","+
60        createJsonPair("tabid", tabID, false) + ","
61        + createJsonPair("cms", assinatura, true) + "}";
62    }
63    catch (std::exception& ex)
64    {
65      throw ex;
66    }
67    return message;
68  }
```

Listing B.3 – Signer.cpp

```
{
  "name": "signerweb",
  "description": "David Grechi Doll",
  "path": "SignerWeb.exe",
  "type": "stdio",
  "allowed_origins": [
    "chrome-extension://dhikfimimcjpoaliefjlffaebdeomeni/"
  ]
}
```

Listing B.4 – host.manifest

ANEXO C – Artigo sobre o TCC no formato SBC

Uma Alternativa aos Applets em Java para Realização de Assinaturas Digitais Através do Navegador Web

David Grehi Döll[1]

[1]Sistemas de informação – Departamento de informática e estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

davidgrdoll@gmail.com

Abstract. *For many years we have used a signature to confirm the validity and express the agreement to certain documentation, currently the handwritten signature has been replaced by digital signatures. Based on the public key cryptography, the digital signature aims to sign electronic documents, guaranteeing the same legal equivalence as the signatures made on paper. Since then, several softwares that have made digital signatures have been developed, and with the increasing use of web browsers and the abandonment of desktop applications, it is becoming increasingly necessary to develop technologies that enable the signature through the browser. With the discontinuation of the Java Applet plugin, some web systems that used it to make signatures, had to look for alternatives for its replacement. This work aims to present solutions to generate signatures through the browser, as well as a functional prototype that solves this problem in the most appropriate way.*

Resumo. *Por muitos anos utilizamos uma assinatura para confirmar a validade e manifestar o acordo à determinada documentação, atualmente a assinatura manuscrita vem sendo substituída por assinaturas digitais. Baseada na criptografia de chaves públicas, a assinatura digital tem como objetivo assinar documentos eletrônicos, garantindo a mesma equivalência jurídica que as assinaturas realizadas no papel. Desde então, diversos softwares que realizam assinaturas digitais vêm sendo desenvolvidos, e com o crescente uso das aplicações Web e o abandono de aplicações desktop, torna-se cada vez mais necessário o desenvolvimento de tecnologias que possibilitem a assinatura através do navegador. Com a descontinuação do plugin Java Applet, alguns sistemas Web que o utilizavam para gerar assinaturas, tiveram que buscar alternativas para a sua substituição. Este trabalho tem como objetivo apresentar soluções para gerar assinaturas através do navegador Web, além de um protótipo funcional que solucione este problema da forma mais adequada.*

1. Introdução

A assinatura digital é utilizada para agregar confiança e segurança às comunicações e negócios vinculados a um ambiente virtual como a Internet, oferecendo eficiência e

rapidez. Além disso, a assinatura digital contribui de forma positiva para o meio ambiente, empresas que armazenam milhares de documentos poderiam digitalizar os mesmos, garantindo a sua validade jurídica através de assinaturas digitais [MENKE 2003].

A assinatura digital é baseada na criptografia assimétrica, que consiste no uso de duas chaves, pública e privada, uma utilizada pelo remetente e outra pelo receptor da mensagem, e é sobre este conceito que se baseia uma assinatura digital. Este par de chaves é gerado por programas de computador ou hardware e as chaves atuam em conjunto. Tudo que é cifrado pela chave pública, só pode ser decifrado pela chave privada correspondente e vice versa. Uma assinatura digital é feita através do uso da chave privada e a verificação é com o uso da chave pública. Somente a chave pública correspondente aquela chave privada poderá interpretar corretamente a assinatura [STALLINGS 2008].

As tecnologias de acesso a chaves criptográficas e outras funções de segurança através de navegadores sempre foram um tópico desafiador. Em um primeiro momento as formas de acessar estas funções eram apenas através de plugins. O navegador deve isolar o ambiente das páginas acessadas do computador por questões de segurança. Podemos separar os plugins feitos para browsers em dois grandes grupos: os especializados e os de uso genérico.

Como exemplos de plugins de uso genérico podemos citar o Flash [ADOBE 2017] e o Java [SMITH 2017]. Exemplos de plugins de uso especializado temos por exemplo o plugin para o skype web [SKYPE 2017]

A primeira arquitetura de plugins utilizada para browsers surgiu com o Netscape. Conhecida como Netscape Plugin API (NPAPI), essa arquitetura foi adotada pela maioria dos browsers dominantes do mercado entre 1995 e 2013. Entretanto, devido a problemas recorrentes de segurança em plugins desenvolvidos com essa API [Oracle 2016], seu uso foi descontinuado [SMEDBERG 2015; Sylvain, Nicolas 2014]. Esta tecnologia possibilita a utilização do plugin Java Applet, que é utilizado para realizar assinaturas através de um navegador.

Devido à descontinuação da mesma, este trabalho tem como objetivo apresentar soluções alternativas para a solução deste problema, através de um protótipo funcional.

2. Criptografia

Criptografia é a escrita de forma ilegível. Cripto, do grego “kryptos”, significa escondido, oculto, e grafia, também do grego “graphos”, significa escrita [PEREIRA 2012].

2.1. Criptografia assimétrica

A criptografia assimétrica utiliza um par de chaves para cifrar e decifrar dados, ou seja, cada chave é utilizada para um propósito diferente [SUTIL et al. 2010].

O conceito de criptografia assimétrica envolve um par de chaves, pública e privada. A chave privada fica sempre em posse do seu dono, e nunca deve ser compartilhada, já a

chave pública é distribuída livremente. Estas chaves são complementares, ou seja, tudo que é cifrado com uma, deve ser decifrado com a outra e vice versa.

2.2. Funções de resumo criptográfico

Funções hash de resumo criptográfico tem um papel fundamental na criptografia. Funções hash recebem uma mensagem como entrada e transformam em uma saída conhecida como hash [ALFRED; PAUL; SCOTT 1996].

Estas funções resumo, também conhecidas como funções hash, são uma operação matemática, que transforma um conjunto de dados em uma sequência de dados de tamanho fixo, geralmente menor do que a sequência original [SUTIL et al. 2010].

2.3. Assinatura digital

Baseada na criptografia de chaves públicas, a assinatura digital permite assinar um documento eletrônico com a mesma validade jurídica de um documento assinado no papel, de forma muito mais segura e promovendo a eliminação de tempo e custos com impressão, transporte, armazenamento e despesas cartorárias. A assinatura digital provê a garantia de integridade, autenticidade e não repúdio de uma mensagem.

Para gerar uma assinatura, deve-se utilizar uma função de resumo para obter o hash do texto plano, e em seguida cifrar este mesmo hash com uma chave privada. Este processo garante a autenticidade da mensagem, uma vez que foi cifrada com uma chave privada e ao decifrar a assinatura com a respectiva chave pública, pode-se comparar o Hash decifrado com o Hash do texto plano original, se forem iguais, a integridade dos dados foi mantida.

2.4. Certificado digital

Determinar quem possui a chave privada é um dos principais problemas na criptografia de chaves públicas. O uso de certificados digitais foi proposto para solucionar este problema. O certificado digital é um conjunto de informações que permitem identificar, de forma segura, o proprietário do par de chaves. Cada certificado contém a chave pública e a identificação da entidade que controla a respectiva chave privada.

2.5. Cryptographic Message Syntax (CMS)

Na prática, uma assinatura digital não contém apenas o resumo criptográfico cifrado. Além do resumo criptográfico, podem ser incluídos também, atributos como data e local onde a assinatura foi realizada, certificado do signatário, sua cadeia de certificação e carimbo do tempo.

Uma assinatura digital pode ser representada pelo formato de assinatura CMS(Cryptographic Message Syntax), atualmente descrito pela RFC 5280, e trata-se de um formato binário, codificado em ASN.1 [SUTIL et al. 2010]. O formato binário ASN.1 é um padrão usado para descrever os dados que são transmitidos entre protocolos de telecomunicações, ou seja, não depende do software e hardware onde foi gerado ou será interpretado [HOUSLEY; POLK 2001].

3. Sandbox dos navegadores Web

Praticamente, o Sandbox(Caixa de areia), é um mecanismo de segurança, utilizado para executar uma aplicação em um ambiente desconhecido. Se um atacante, consegue de alguma forma, se aproveitar de uma falha de segurança para executar um código malicioso na máquina, este modo impede que o dano seja causado no sistema, além de prevenir a modificação e leitura de arquivos do mesmo. Na grande maioria dos navegadores, toda a parte de renderização HTML e Javascript é isolada em um processo dentro deste módulo.

4. Tecnologias que possibilitam a assinatura através do navegador Web

4.1. Applets Java

As applets em Java são uma maneira de incluir código nativo e executar nos clientes que visualizam uma página web. O plugin NPAPI, tecnologia que possibilitava o uso dos Applets em Java nos navegadores Web foi descontinuado, tornando esta abordagem obsoleta.

4.2. ActiveX

ActiveX é um framework que já está se tornando obsoleto. Criado em 1996 pela Microsoft para a plataforma Windows. Controles ActiveX podem ser incluídos dentro de páginas web para realizar diversas ações, inclusive executar código nativo, similar aos Applets em Java.

4.3. Websockets

Para reduzir o retardo nos serviços como o protocolo HTTP, surge o Websocket. WebSockets são um padrão ainda em desenvolvimento e evolução sob os cuidados da IETF.

Uma API padrão para implementação deste protocolo está sendo formalizada pela W3C para que os navegadores ofereçam suporte ao mesmo [ESPOSITO 2017]. Este protocolo tem como objetivo superar as limitações estruturais do protocolo HTTP, que é considerado ineficiente quando utilizado em aplicações que precisam permanecer conectadas com o servidor em uma conexão persistente. Por meio de um soquete TCP, este protocolo permite uma comunicação bidirecional entre os aplicativos e servidores Web.

4.4. Web Extensions e Native Messaging

Extensões são complementos adicionados ao navegador, e que podem modificar a funcionalidade do mesmo. Estes complementos são escritos utilizando as tecnologias padrões da Web, como Javascript, HTML e CSS.

As extensões para o navegador Web Firefox, são construídas utilizando a API WebExtensions, esta API proporciona um sistema multi-navegador para desenvolvimento de extensões. Através desta API é possível criar uma extensão que seja compatível com o navegador Google Chrome e Opera, além disso, extensões

escritas para estes navegadores, irão funcionar no Microsoft Edge com poucas mudanças [FTHIAGOGV; MOSTARD 2017].

Através das extensões, é possível utilizar Native Messaging. Native Messaging é uma tecnologia que permite que a extensão troque mensagens com uma aplicação nativa instalada no computador de um usuário. Um exemplo comum são os gerenciadores de senha (Password managers), que utilizam Native Messaging para gerenciar os Stores(Gerenciadores de senha e certificados) e a criptografia das senhas.

4.5. Web Crypto API

Web Crypto API é uma tecnologia experimental que vem sendo desenvolvida pelos navegadores atuais, esta API apresenta uma interface de acesso a funções criptográficas primitivas para criar sistemas que utilizam criptografia. Trata-se de uma abordagem nova, que permite até mesmo a realização de assinaturas e verificação da mesma. Entretanto, ainda que esta tecnologia possibilite estas operações, a Web Crypto API é incapaz de interagir com hardwares dedicados, como cartões inteligentes (smartcards), tokens USB ou geradores de aleatoriedade [GPRIMOLA HIKASHII 2018].

5. Trabalhos relacionados

Além das tecnologias citadas na seção anterior, foram revisados alguns trabalhos que fizeram uso das mesmas para resolver problemas semelhantes. Como exemplo podemos citar o trabalho do autor [TRAPP 2016], que na busca por uma alternativa aos applets para acesso a dispositivos, realizou uma implementação que utiliza um servidor HTTP local. O autor sugere como trabalhos futuros, o uso do protocolo de comunicação HTTPS, e também a implementação utilizando Websockets. Já o autor [SOARES 2017], implementou um sistema de emissão de certificados online utilizando um servidor local HTTPS.

Estes dois trabalhos citam as dificuldades em relação a utilização de um servidor local, um deles é a necessidade de instalação do certificado da conexão SSL, que gera diversos problemas de segurança com os navegadores web e programas antivírus. Ambos sugerem uma solução que não exija a instalação de um certificado.

6. Proposta e desenvolvimento do protótipo de assinaturas digitais através do navegador web

As Extensões Web, ou Web Extensions, são complementos que podem ser adicionados no navegador do usuário, possibilitando a comunicação com um módulo nativo que realiza assinaturas. Através de um instalador, este módulo nativo deve ser registrado no sistema, e uma vez que foi registrado, possibilita que o navegador se comunique com o mesmo através da linguagem Javascript.

Diferente dos WebSockets, um processo só é aberto quando uma chamada é feita, e removido somente quando a requisição é finalizada. As extensões Web baseadas na API WebExtensions, são compatíveis entre diversos navegadores, por exemplo, uma extensão desenvolvida para o navegador Google Chrome, pode ser utilizada nos

navegadores Firefox e Opera praticamente sem que alterações precisem ser feitas. Para este trabalho, a tecnologia Web Extensions foi escolhida por demonstrar ser mais simples de ser desenvolvida e não apresentar os problemas encontrados nas tecnologias como Applets, WebSockets e ActiveX. Extensões podem ser facilmente instaladas no computador do usuário, não apresentando problemas relacionados à antivírus e firewall por exemplo.

O formato CMS (Cryptographic Message Syntax) foi escolhido como modelo de assinatura por ser mais simples de desenvolver, porém, qualquer formato de assinatura existente poderia ter sido escolhido. Portanto, o modelo de proposta deste trabalho consiste em uma solução que realize uma assinatura digital CMS através de uma extensão Web para o navegador Google Chrome no sistema operacional Windows.

6.1. Content script e Background script

Toda extensão possui um arquivo manifest.json, e este é o único arquivo que obrigatoriamente deve estar presente em toda extensão. Nele estão informações muito importantes, como nome da extensão, versão, permissões utilizadas e ponteiros para outros arquivos que também serão incluídos e utilizados no pacote da aplicação [GOOGLE 2018]. Dois tipos de arquivos muito importantes incluídos no manifest.json são os Content scripts e Background scripts.

Content scripts fazem parte da estrutura de uma extensão, e são incluídos no seu manifest. Escritos na linguagem Javascript, podem ser utilizados para interagir com as páginas Web. Este script é executado no contexto de uma página que foi carregada pelo navegador, como se fosse uma parte dela. Através do content script é possível ler detalhes das páginas Web visitadas pelo navegador, escutar eventos, modificar o conteúdo ou layout, e até mesmo comunicar-se com outras extensões [GOOGLE 2018].

Através do background script é possível realizar chamadas que nenhum outro script teria permissão, isto porque ele é executado em uma camada de baixo nível no navegador, tornando possível por exemplo, enviar mensagens nativas, funcionalidade essencial para o desenvolvimento deste trabalho.

6.2. Native Messaging

Extensões podem trocar mensagens com aplicações nativas utilizando Native Messaging. Primeiramente, a página HTML que está carregada no browser irá se comunicar com o content script da extensão, solicitando uma funcionalidade que utiliza Native Messaging, por exemplo, uma assinatura. Como apenas o background script é capaz de realizar chamadas nativas, esta solicitação será apenas repassada para ele. O background script irá receber esta solicitação e tentar se conectar a aplicação nativa, vamos chamar esta aplicação nativa de host. Para que o background script possa realizar esta conexão, é necessário que o host esteja registrado no sistema.

No caso do Windows, precisamos criar uma nova chave nos registros, e o valor desta chave deve apontar para o caminho de um arquivo manifest que contém todas as informações deste host. O fato de registrar o host no sistema, faz com que ele se torne visível para o navegador. Desta forma, o background script irá conseguir se conectar à

aplicação, isto porque o valor do registro deve apontar para um arquivo chamado host manifest, e este arquivo contém o caminho para a aplicação instalada.

7. Protótipo funcional

Esta seção apresenta uma descrição de como foi desenvolvido o protótipo funcional deste trabalho, que será uma extensão para o navegador Google Chrome. Esta extensão será chamada de Signer Web.

7.1. Content script e background script

Foram incluídas duas funções no content script da extensão, estas funções escutam os eventos sign e list_certificates. Quando a extensão receber um destes eventos, irá repassar a mensagem para o background script, através da função Javascript chrome.runtime.sendMessage, esta função é utilizada para realizar a comunicação entre a extensão e o background. No background script, caso o action recebido seja list_certificates ou sign, envia uma mensagem nativa para o host instalado no computador, através da função chrome.runtime. sendNativeMessage. Esta função é responsável por realizar a comunicação nativa com o host.

7.2. Função de callback

Sabemos que todo este processo deu início em uma página HTML qualquer que disparou um evento para a extensão solicitando uma ação. Este evento foi tratado pelo content script, e repassado para o background script, que por sua vez enviou a mensagem para a aplicação nativa (host) instalado na máquina do usuário. A mensagem foi processada pelo host e retornada para o background script, e o mesmo enviou a mensagem de volta para o content script.

Agora que a mensagem foi processada, o content script precisa responder a chamada desta página HTML, supondo que esta página queira exibir uma lista com os certificados instalados na máquina do cliente, o content script precisa enviar esta lista para a página. Neste trabalho isso foi feito através de uma função de callback.

Definindo o nome da função de callback de forma constante, a página que desejar tratar o retorno do content script, precisa ter esta função declarada com o mesmo nome no seu código Javascript, desta forma o content script irá chamar esta função enviando a solicitação processada como parâmetro. Isto pode ser feito porque o content script está lado a lado com o script da página, porém ele não tem como saber qual é a função de callback que será utilizada, por isso ela é definida de forma estática, e aqueles que quiserem receber o retorno, precisam declará-la com este nome.

7.3. Aplicação nativa host

Quando o background script envia uma mensagem nativa através da função chrome. runtime.sendNativeMessage, o Google Chrome irá buscar nos registros do Windows, pelo identificador incluído no primeiro parâmetro da função.

Neste trabalho, o host foi desenvolvido na linguagem C++, por questões de afinidade com a linguagem, e performance, porém poderia ter sido desenvolvido em outra linguagem, como Java ou Pyhton por exemplo. Todo o procedimento de registro e instalação da aplicação foi feito através de um instalador MSI. Este instalador é responsável por copiar os arquivos host e host manifest para um diretório específico, e criar uma chave nos registros que aponte para o arquivo host manifest.

Agora que o host já pode ser encontrado pela função chrome.runtime.sendNativeMessage, ele deve ser capaz de processar as solicitações enviadas pela extensão, e respondê-las.

8 Considerações finais

Através do protótipo desenvolvido, podemos perceber como fica fácil para uma aplicação Web qualquer utilizar esta extensão, basta solicitar que os seus usuários instalem a extensão e a aplicação nativa, e já será possível comuicar-se para gerar assinaturas.

Por ser uma comunicação entre processos, também evitamos problemas com antivírus. Um dos problemas relacionados ao WebSocket e servidores locais, é que muitas vezes a conexão não é segura, e quando é, o anti-vírus suspeita do que está sendo enviado e recebido, por este motivo muitas vezes impede a aplicação de funcionar.

Diferente dos servidores locais e Websockets, a comunicação da extensão com a aplicação nativa é feita através de processos locais que executam no computador do usuário, e não serviços. Apesar de não utilizar protocolos seguros como o SSL, podemos dizer que esta comunicação é segura, e que apenas um programa malicioso com privilégios de administrador conseguiria interceptar a troca de mensagens entre a extensão e a aplicação nativa. Portanto, a não ser que o usuário instale algum software malicioso no seu computador, não teria como tornar esta comunicação insegura. Além disso, foi definido no host manifest, que apenas a extensão Signer Web pode trocar mensagens com o host, sendo assim, não existe a possibilidade de outra extensão tentar se conectar a ele.

Outra questão importante, é que o processo de comunicação entre a extensão e o host, só é executado quando uma requisição é realizada, e após a finalização o processo é deletado.

9 Conclusão

Este trabalho teve como problema inicial a descontinuação da tecnologia Java Applet que possibilitava o acesso a dispositivos criptográficos através de um browser. O fim desta tecnologia criou a necessidade de busca por uma nova forma de realizar assinatura através dos navegadores Web.

As tecnologias existentes que possibilitam a solução deste problema são, o Java Applet, componentes ActiveX, WebSockets e WebExtensions. Por questões de usabilidade, segurança e facilidade de desenvolvimento, a tecnologia WebExtension foi escolhida, e um protótipo que atende aos objetivos específicos deste trabalho foi desenvolvido.

Podemos confirmar que extensões podem ser utilizadas para realizar assinaturas e acessar dispositivos criptográficos através de um navegador web, utilizando a tecnologia Native Messaging dos navegadores. Estas extensões podem ser facilmente integradas a qualquer aplicação web, de forma segura.

Através do desenvolvimento deste protótipo, capaz de realizar assinaturas CMS e listar os certificados instalados no computador do usuário ou dispositivos criptográficos, podemos concluir que as extensões são uma ótima forma de resolver este problema. Além de vantagens de usabilidade, podemos citar diversas outras, como por exemplo, a segurança, solução de problemas relacionados a programas antivírus e a possibilidade de instalação sem permissão de administrador, que eram os principais objetivos deste trabalho.

A extensão que soluciona o problema apresentado por este trabalho, foi implementada com foco no sistema operacional Windows, e para o navegador Google Chrome. A API de desenvolvimento de extensões WebExtension's, possibilita que uma mesma extensão possa ser compátivel com diversos navegadores, como o Firefox, Edge e opera. Bastaria que alguns pequenos ajustes fossem realizados.

Como tema para trabalhos futuros, seria interessante o desenvolvimento de uma extensão que fosse compatível com estes navegadores e outros sistemas operacionais. Como o módulo nativo foi escrito em C++, o código poderia ser aproveitado, tornando necessário apenas alguns ajustes em relação a forma comos os certificados são acessados em outros sistemas operacionais.

10. References

ADOBE. Adobe Flash Player, http://get.adobe.com/br/flashplayer/about/, 2017.

ALFRED, J. M.; PAUL, C. v. O.; SCOTT, A. V. Handbook of applied cryptography. Massachusetts Institute of Technology, p. 560, 1996.

ESPOSITO, D. Cutting Edge: Compreendendo o poder dos WebSockets, https://msdn.microsoft.com/pt-br/magazine/hh975342.aspx, 2017.

FTHIAGOGV; MOSTARD. O que são extensões? https://developer-.mozilla.org/pt-BR/Add-ons/WebExtensions/What, 2017.

GOOGLE. Getting Started Tutorial, https://developer.chrome-.com/extensions, 2018.

GPRIMOLA HIKASHII, c. Web Crypto API. 2018.

HOUSLEY, R.; POLK, T. Planning for PKI: best practices guide for deploying public key infrastructure. [S.l.]: John Wiley & Sons, Inc., 2001.

MENKE, F. Assinaturas digitais, certificados digitais, infra-estrutura de chaves públicas brasileira e a icp alemã. Revista de Direito do Consumidor, v. 12, n. 48, 2003.

ORACLE. Getting Started With Applets, https://docs.oracle.com/javase-/tutorial/deployment/applet/getStarted.html, 2014.

PEREIRA, A. S. d. S. Aplicabilidade do padrão de assinatura pades no Âmbito da icp-brasil. Universidade Federal de Santa Catarina, 2012.

SKYPE. O que é o Skype Web Plugin e como faço para instalá-lo? https://support.skype.com/pt-br/faq/FA12316/o-que-e-o-skype-web-plugin-e-como-faco-para-instala-lo, 2017.

SMEDBERG, B. NPAPI Plugins in Firefox, https://blog.mozilla.org-/futurereleases/2015/10/08/npapi-plugins-in-firefox/, 2015.

SMITH, D. JDK 9 e o Plug-in Java, https://www.java.com-/pt, 2017.

SOARES, P. Alternativa para emissão de certificados por autoridade certificadora online. Universidade Federal de Santa Catarina, 2017.

STALLINGS, W. Criptografia e segurança de redes: princípios e práticas. [S.l.]: Pearson Prentice Hall, 2008.

SUTIL, J. M. et al. Introdução a Infraestrutura de Chaves Públicas e Aplicações (ICPEDU). [S.l.]: Escola Superior de Redes RNP, 2010.

TRAPP, M. Uma alternativa aos applets em java para acesso a dispositivos. Universidade Federal de Santa Catarina, 2016.