Aplicativo para verificação de conformidade de assinaturas digitais no âmbito da ICP-Brasil

Gustavo José Carpeggiani

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
R. Eng. Agronômico Andrei Cristian Ferreira, s/n - Trindade, Florianópolis - SC, 88040-900, Brasil

Abstract. *Digital signatures are increasingly present in Brazilian services, such as in the new national digital drivers license, but there is still no portable application that can verify a digital signature in accordance with the Brazilian regulatory standards. Hence, this work aims to meet this need, providing convenient access on mobile platforms for verification of compliance in digital signatures in Brazil. The result obtained, was an application built with Ionic Framework, which, using a single generic code base, generates specific code for the specific build platforms. The resulting application, was tested primarily on Android, and is able to verify signatures according to the national public key infrastructure standards.*

Resumo. *Assinaturas digitais estão cada vez mais presentes nos serviços do brasileiro, como na nova carteira nacional de habilitação digital, mas ainda não existe um aplicativo verificador de assinaturas digitais portátil de acordo com as normas brasileiras estabelecidas pelo Instituto Nacional de Tecnologia da Informação (ITI). Desta maneira este trabalho visa suprir esta necessidade, providenciando acesso conveniente em plataformas móveis para a verificação de conformidade em assinaturas digitais no Brasil. O resultado obtido foi uma aplicação construída com o Ionic Framework, que a partir de uma única base de código genérico, gera código específico para a plataforma alvo de compilação. A aplicação resultante foi testada primariamente em Android, e é capaz de verificar assinaturas de acordo com as normas da infraestrutura de chaves públicas nacional.*

1. Introdução

Os processos de comunicação evoluem rapidamente, possibilitando um grande volume de informações serem enviadas e recebidas. Consequentemente, isto ocasiona a necessidade de tratar tais informações de forma rápida e segura. Para que tal comunicação não fique vulnerável a alterações ou roubo de informação, os mecanismos de autenticação e segurança também avançam de maneira rápida para atender estas necessidades.

Atualmente a comunicação digital ocorre de forma análoga à comunicação natural, e para preservar a privacidade destas informações que trafegam no meio digital, são utilizados vários conceitos de criptografia. Um dos mais importantes conceitos para a criptografia é a assinatura digital, que garante ao receptor de uma mensagem assinada, a verificação da legitimidade mensagem recebida por meio de chaves criptográficas.

De acordo as divulgações digitais em [Exame 2018], [Oliveira 2018] e [Wakka 2018], é evidente o fato de que as assinaturas digitais estão cada vez mais presentes no cotidiano do brasileiro, seja em documentos importantes, como a Carteira Nacional de Habilitação digital, ou no uso empresarial, como no transporte de mercadorias

em serviços de logística. Além do uso pessoal para troca de mensagens e arquivos, entre outros usos.

A principal tarefa a ser realizada em softwares utilizados pelos usuários de assinaturas digitais é a verificação das assinaturas, e no Brasil, ainda não existe um verificador de assinaturas digitais na forma de aplicativo móvel em conformidade com os padrões nacionais de assinaturas digitais.

1.1. Justificativa

Atualmente existe um verificador para assinaturas digitais no âmbito da Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil, disponibilizado em [ITI 2018], pelo Instituto Nacional de Tecnologia da Informação (ITI), mas este foi projetado para *browsers* da *web*, o qual não é perfeitamente adequado para uso em celulares e *tablets*. Segundo o relatório divulgado por [Higa 2018], a maioria dos usuários da *web* se concentra em dispositivos móveis, e portanto, é notória a existência da necessidade de atender as necessidades destes usuários com uma aplicação destinada às suas plataformas específicas para que seja propiciada uma experiência ideal ao usuário, com a adição de melhorias de acesso.

1.2. Objetivos

Implementar um aplicativo móvel que possua a funcionalidade de verificar assinaturas digitais no padrão brasileiro de assinaturas digitais, de acordo com as normas estabelecidas no DOC-ICP-15 da ICP-Brasil, para que um dado usuário possa ter sempre presente, uma maneira rápida e eficiente de verificar as assinaturas de seus arquivos onde quer que esteja, de forma conveniente.

1.3. Objetivos Específicos

- Realizar a implementação de um aplicativo multiplataforma que verifique assinaturas digitais em conformidade com as normas do DOC-ICP-15.
- Providenciar ao usuário final um aplicativo simples e objetivo. Que resolva as necessidades na plataforma móvel de escolha dele.
- Documentar o processo de implementação deste aplicativo, para que outros trabalhos futuros possam se beneficiar dos conhecimentos adquiridos durante a implementação.

1.4. Metodologia

Foi realizado um estudo sobre criptografia e conceitos básicos que envolvem o modelo de assinaturas digitais: criptografia de chave assimétrica e funções de resumo criptográfico. Após o estudo dos conceitos básicos, foram estudados os manuais das ferramentas utilizadas para a implementação do software móvel. Com a pesquisa de material concluída, o aplicativo foi devidamente implementado, testado e documentado. O desenvolvimento foi realizado, resultando em uma única base de código genérico que pode ser compilado para diversas plataformas utilizando *Apache Cordova*.

1.5. Organização dos Capítulos

Os capítulos deste artigo estão organizados na seguinte ordem:

- Introdução: Apresentação do problema e os motivos relacionados ao mesmo, juntamente da justificativa para a realização do trabalho

- Fundamentação Teórica: Apresentação dos conceitos básicos envolvidos no âmbito deste trabalho.
- Verificador de conformidade de assinatura digital: Apresentação e detalhamento da solução para o problema, com base no que foi aprendido com a pesquisa teórica.
- Ferramentas e Tecnologias: Um detalhamento das ferramentas e tecnologias que foram utilizadas para realizar a proposta.
- Implementação: Detalhes da execução das tarefas realizadas para a implementação da proposta.
- Conclusão: A apresentação dos principais pontos de aprendizado e dos resultados obtidos.

2. Fundamentação Teórica

Neste capítulo serão apresentados os conceitos teóricos básicos para a realização deste trabalho.

2.1. Criptografia

De acordo com [Anderson 2010], a criptografia é o campo em que a Engenharia de Segurança se encontra com a Matemática. Ela nos providencia as ferramentas que hoje em dia estão presentes nos mais modernos sistemas de segurança, e isto possibilitou a criação de sistemas distribuídos com controle de acesso seguro e consequentemente de toda a *internet*. A principal funcionalidade da criptografia é o ciframento de uma mensagem pelo seu autor, para que ele possa enviá-la ao destinatário de forma segura, sem que um terceiro possa acessar ou alterar a mensagem durante sua passagem pelo seu meio de transporte.

Segundo [Katz and Lindell 2007], os três princípios básicos que envolvem um processo de criptografia moderno são os seguintes:

- Princípio 1: O primeiro passo para explicar um problema criptográfico é a formulação de uma rigorosa e precisa definição de segurança.
- Princípio 2: Quando a segurança de uma construção criptográfica se basear em uma suposição não provada, esta suposição deve ser precisamente declarada. Além disso, a suposição deve ser a mínima possível.
- Princípio 3: Construções criptográficas devem ser acompanhadas de provas de segurança rigorosas com respeito a uma definição formulada de acordo com o princípio 1, e relativa a uma suposição declarada como no princípio 2 (caso uma suposição seja necessária).

Tais princípios são fundamentais para o estabelecimento de um conjunto de regras formais que garantam o funcionamento adequado dos sistemas criptográficos modernos. Em relação ao escopo deste trabalho, será abordado o sistema de criptografia de chave assimétrica, pois é o sistema empregado nas assinaturas digitais.

2.2. Criptografia de Chave Assimétrica

Uma chave, é um trecho de informação digital inserida como parâmetro de uma função criptográfica, que determina sua saída, com base no seu conteúdo. É usada para transformar um texto simples em um texto cifrado e vice-versa.

Em um sistema criptográfico assimétrico as chaves são organizadas em pares, sendo uma destas chaves pública e outra privada. Um exemplo do uso deste modelo seria um indivíduo que publica em uma página na *web* sua chave pública com a qual terceiros podem criptografar mensagens para enviar para ele. Depois o proprietário da página da *web* pode descriptografar a mensagem usando a chave privada correspondente.

Como afirmado por [Anderson 2010], uma das principais aplicações da criptografia assimétrica é a assinatura digital. A ideia é de que se pode assinar uma mensagem usando uma chave privada de assinatura, e depois qualquer pessoa pode verificar a autenticidade e integridade do documento usando a chave pública disponibilizada pelo dono do par de chaves, para atestar o não-repúdio da mensagem. A autenticidade é confirmada devido ao fato que, apenas alguém com a chave privada poderia ter criado a assinatura, que é verificada usando a chave pública do par, a integridade é garantida pela função de resumo criptográfico, e o não-repúdio é aceito, caso todas as informações estejam de acordo com a prova de origem.

Um modelo de comunicação que usa criptografia de chave pública, de acordo com [Stallings 2005] possui os seguintes elementos:

- Uma mensagem em texto simples.
- Um algoritmo para cifrar a mensagem.
- Um par de chaves: uma pública e uma privada.
- Texto cifrado, resultado da criptografia da mensagem em texto plano usando o algoritmo de cifragem.
- Um algoritmo para decifrar o texto cifra.

Em sequência, os seguintes passos são essenciais para a realização da comunicação entre duas partes:

- Primeiro, cada parte gera um par de chaves que serão usados para cifrar e decifrar as mensagens.
- Cada usuário coloca uma das chaves do par em um registro público para o outro acessar. Esta será a chave pública. A outra chave deve ser guardada pelo usuário, denominada chave privada.
- Se João deseja enviar uma mensagem confidencial para Maria, João cifra a mensagem usando a chave pública de Maria.
- Quando Maria recebe a mensagem, ela decifra a mensagem usando sua chave privada. Nenhum outro indivíduo pode decifrar a mensagem, pois apenas Maria possui a chave privada.

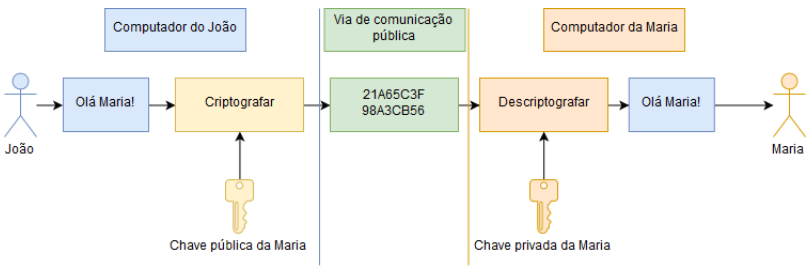

Figura 1. Esquema de comunicação com modelo de chave assimétrica.

2.3. Funções de Resumo Criptográfico

De acordo com [Anderson 2010], funções de resumo criptográfico, também conhecidas como funções de *hash* criptográfico, foram as primeiramente usadas em sistemas de computador para criptografia unidirecional de senhas nos anos 60, e são usadas até hoje em vários sistemas de autenticação.

São utilizadas para verificar a integridade de arquivos, pois em um arquivo corrompido ocorrerá alteração dos bits, que serão detectados observando o *hash*. Em outro caso de uso, se alguém procura provas de que se possui um determinado documento eletrônico até uma determinada data, é possível submetê-lo a um serviço de carimbo de hora, que utiliza *hash* para verificar isto.

Em aplicações de mensagens, os *hashes* são muitas vezes conhecidos como *digests* (ou resumos) das mensagens. Dada uma mensagem M pode-se passá-la através de uma função para obter um resumo, digamos $h(M)$, que pode substituir a mensagem em várias aplicações. Isto é um conceito chave para a assinaturas digitais, pois de acordo com [Anderson 2010] algoritmos de assinatura tendem a ser lentos se a mensagem for muito extensa, então é conveniente assinar um resumo de mensagem ao invés da mensagem completa.

De acordo com [Stallings 2005] uma função de *hash* possui a seguinte forma:

Um valor $hash$ é gerado por um função $h(M)$ da forma $hash = h(M)$ onde M é a mensagem de tamanho variável e $h(M)$ é o valor de *hash* de tamanho fixo. O valor $hash$ é então acrescentado à mensagem na fonte no tempo em que a mensagem é presumida ou confirmada ser correta. O receptor da mensagem autentica a mensagem recalculando o valor $hash$. Com isso em mente, pode-se notar o principal propósito destas funções: **realizar um mapeamento de dados de tamanho arbitrário para um tamanho fixo**.

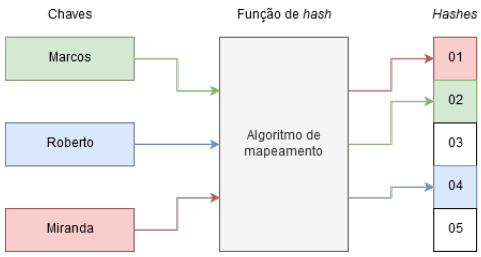

Figura 2. Exemplo de uma função de *hash* simples que identifica o nome de uma pessoa.

O propósito das funções de *hash* criptográfico, é produzir uma espécie de impressão digital de um arquivo, mensagem ou bloco de dados. Segundo [Stallings 2005], a função de *hash*, $hash = h(M)$, deve possuir as seguintes propriedades para que funcione adequadamente:

- h pode ser aplicado em um bloco de dados de qualquer tamanho.
- h produz uma saída de tamanho fixo.
- $h(x)$ é relativamente fácil de computar para qualquer x, fazendo tanto implementações em software como em hardware fáceis de realizar.
- Para qualquer dado valor de $hash$, é computacionalmente inviável, encontrar um x dado que $h(x) = hash$. Esta é denominada a propriedade *one-way*.

- Propriedade de resistência à colisão fraca: Funções de *hash* para qualquer dado bloco x, é computacionalmente inviável encontrar xy tal que $h(y) = h(x)$.
- Propriedade de resistência à colisão forte: É computacionalmente inviável achar qualquer par (x, y) tal que $h(x) = h(y)$.

2.4. Assinaturas Digitais

Uma assinatura é uma marca que alguém coloca em um documento para garantir que o mesmo foi autenticado por quem assinou. Em um documento físico, a garantia de segurança que uma assinatura proporciona é limitada pela verificação visual da mesma, ou seja, está sujeita a caligrafia manual de uma certa pessoa, a qual é supostamente difícil de ser copiada [Anderson 2010]. Devido a ser uma maneira relativamente instável de verificação, pois normalmente o receptor da mensagem não tem conhecimento da assinatura devido a falta de padronização, as instituições humanas começaram a adotar outras formas de assinatura, em que ambas as partes, receptora ou emissora de um dado documento, pudessem ter uma verificação estável na autenticação, o que ocasionou no uso de selos, e posteriormente de carimbos. Atualmente este modelo de assinaturas é aplicado de forma similar no meio digital para autenticação, verificação de integridade e não-repúdio de arquivos e mensagens, com a ajuda de conceitos criptográficos.

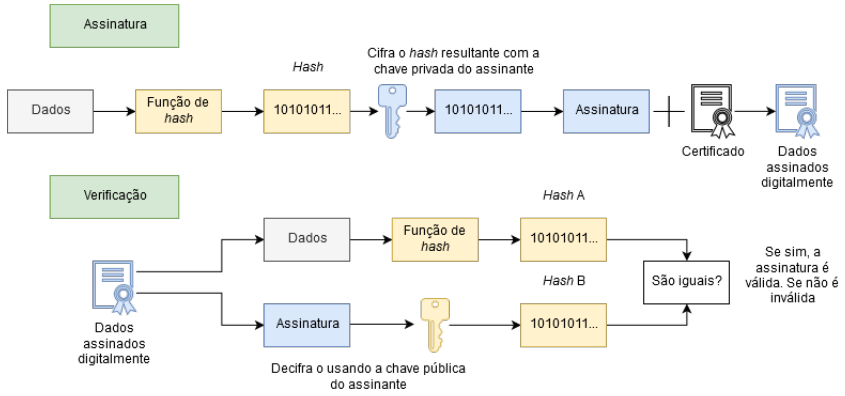

Figura 3. Exemplo de criação e verificação de uma assinatura digital

De acordo com [Anderson 2010], assinaturas digitais possuem três principais processos associados: a geração de chaves para a assinatura, a realização da assinatura pelo emissor da mensagem, e a verificação da assinatura pelo receptor. Esquemas de assinatura podem ser *determinísticos* ou *aleatórios*, no primeiro caso sempre resultando na mesma assinatura, enquanto no segundo gera-se sempre um novo resultado, sendo assim mais associado com as assinaturas feitas à mão, pois nenhuma se iguala à outra, mas ao mesmo tempo pode-se verificar se ela é forjada ou autêntica. No escopo deste trabalho serão observados apenas os esquemas *determinísticos*.

Formalmente, um esquema de assinatura, assim como um esquema de criptografia com chave pública, possui uma função geradora de par de chaves criptográficas em que uma entrada aleatória *R* sempre retornará duas chaves, a chave privada *SK* e a chave pública *PK*. Tais chaves possuem as seguintes propriedades:

- Dada uma chave pública de verificação de assinatura *PK*, é inviável computar a chave privada de assinatura *SK*.

- Existe uma função de assinatura digital em que dada uma mensagem *M* e uma chave de assinatura privada *SK*, irá produzir a assinatura *SigSK(M)*.
- Existe uma função de verificação de assinatura que dada uma assinatura *SigSK(M)* e a chave de verificação de assinatura pública *PK* retornará o valor lógico VERDADEIRO se a assinatura foi computada corretamente com *SK* e a mensagem não foi alterada, e caso contrário retorna o valor lógico FALSO.

Um simples algoritmo de assinatura digital pode ser modelado como sendo uma função aleatória que reduz qualquer mensagem de entrada para um *hash one-way* de tamanho fixo, seguida por uma cifra de bloco que realiza a operação em uma direção, denominada assinatura, para apenas um indivíduo, o dono da chave privada. Enquanto na outra direção é permitido que qualquer indivíduo realize o processo de verificação.

O processo de verificação de assinatura pode ser realizado de maneira simples, em um esquema básico, o algoritmo de verificação de assinatura emite na saída valores lógicos VERDADEIRO ou FALSO, dependendo se a assinatura é adequada. Mas outros processos existem, em que é possível um esquema com recuperação de mensagem, em que qualquer indivíduo pode inserir uma assinatura e receber de volta a mensagem correspondente a ela. Isto significa que no ponto de vista do algoritmo, ele já viu esta assinatura previamente e possui uma mensagem associada a ela, caso contrário ele irá associar um valor aleatório e salvar esta entrada e a saída aleatória como um par de assinatura e mensagem correspondentes. No entanto, genericamente falando, não há a necessidade de recuperação de mensagem, pois a mensagem ao ser assinada possui um comprimento arbitrário e será enviada para uma função de *hash* e, em seguida, assinar o valor de *hash*.

Devido a natureza do funcionamento das assinaturas digitais, as chaves públicas precisam ser divulgadas de uma maneira ampla e disponível, sendo assim, é criada a necessidade de um sistema que organize e forneça acesso eficiente para os usuários, as assinaturas públicas existentes e maneira transparente e segura. Para isto são necessárias as infraestruturas de chaves públicas.

2.5. Infraestrutura de Chaves Públicas

Em decorrência da necessidade do estabelecimento de confiança durante o processo de comunicação em dispositivos digitais, foram criadas as infraestruturas de chaves públicas (ICP's), do inglês PKI *(public key infrastructure)*.

De acordo com [Choudhury 2002], uma infraestrutura de chaves públicas é um *framework* composto por *hardware*, *software*, políticas e procedimentos para gerenciar chaves e certificados. Para que este *framework* seja funcional, são necessários vários componentes que trabalham em união para realizar seus serviços.

2.6. Autoridade de certificação

Uma autoridade de certificação (AC), uma entidade confiável que autentica entidades tomando parte em uma transação eletrônica. Para autenticar uma entidade, a AC emite um certificado digital. Antes da emitir um certificado digital a AC verifica a requisição do mesmo em uma autoridade de registro. Caso a requisição feita seja validada, o certificado é emitido.

2.7. Autoridade de registro

Uma autoridade de registro (AR), é responsável pela interação entre clientes e AC's. Frequentemente, devido a alta quantidade de requisições de certificados, não é possível que a AC aceite requisições de certificados, valide tais requisições e emita os certificados. Para atender estes casos, a autoridade de registro atua como intermediária entre a AC e o cliente. Suas tarefas englobam, receber requisições de entidades e validá-las, enviar requisições para a AC, receber o certificado processado pela AC e enviar o certificado para a entidade adequada.

2.8. Clientes de ICP

Os clientes da Infraestrutura Chaves Públicas, são as entidades que realizam requisições para a AC ou AR para emissão de certificados. Para obter um certificado digital de uma AC, um cliente precisa enviar uma requisição para gerar um par de chaves pública e privada, tal par de chaves contém os detalhes do cliente. Com o par de chaves gerado, a requisição de certificado é então enviada a AC, podendo ser desviada para uma AR. Após isto, o cliente receberá o certificado da AC, e pode usá-lo para identificar-se como sendo um portador de certificado autenticado na infraestrutura.

2.9. Certificados digitais

Certificados digitais, são mecanismos de integridade de dados, que são usados pelas AC's para garantir a autenticidade das chaves na infraestrutura. Eles vinculam a chave pública e suas informações associadas com o dono de uma maneira confiável. Garantindo que apenas a chave pública de um certificado que foi autenticado por uma AC funcione com a chave privada em posse de uma entidade. Isto elimina as chances de personificação dentro da infraestrutura e garante a segurança. Os principais elementos de um certificado digital são, o seu número serial, a assinatura digital da AC, a chave pública do usuário emissor do certificado, data de validade, nome da AC emissora.

2.10. Repositório de certificados digitais

O repositório de certificados digitais, é responsável pela distribuição de certificados para usuários e organizações. Estes certificados podem ser distribuídos em duas maneiras, de acordo com a implementação da organização da ICP. Os certificados podem ser distribuídos pelos próprios usuários ou podem ser distribuídos por um servidor do repositório. Entre as tarefas que são realizadas se encontram, gerar e emitir pares de chaves, certificar a validade de chaves públicas assinando a chave pública, revogar chaves expiradas ou perdidas e publicar chaves públicas no servidor de serviço.

2.11. ITI

O Instituto Nacional de Tecnologia da Informação (ITI)[1], é um órgão do governo, associado a Casa Civil da Presidência da República, que tem por missão manter e executar as políticas da Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil). Ao ITI compete ainda ser a primeira autoridade da cadeia de certificação digital – AC Raiz. Compete ao ITI coordenar a ICP-Brasil para que fique de acordo com as regulamentações do DOC-ICP-15. [ITI 2017b]

[1] http://www.iti.gov.br/

2.12. ICP-Brasil

É a a infraestrutura de chaves públicas brasileira, regida pelo ITI, seguindo as normas estabelecidas no DOC-ICP-15. De acordo com [ITI 2017a], a ICP-Brasil possui os seguintes entes relacionados:

- Autoridade Certificadora Raiz (ACR), responsável pela gerência dos certificados digitais.
- Autoridade Certificadora (AC), são intermediárias emitidas pela ACR, responsáveis pela gerência de certificados digitais, sendo diretamente subordinadas à ACR. Podem emitir outras AC's, e AC's denominadas "finais"emitem certificados para usuários finais.
- Autoridade de Registro (AR), é responsável pela intermediação entre o usuário da ICP-Brasil e as AC's. Podendo estar fisicamente associada com alguma AC.
- Autoridade Certificadora do Tempo (ACT), é responsável por emissões de *timestamps*, ou carimbos de tempo, que validam as questões temporais de uma transação e seus componentes associados.

Existem ainda os entes de Prestador de Serviço de Suporte e Prestador de Serviço Biométrico, mas não fazem parte do escopo deste trabalho e não serão detalhados.

Como se pode observar, esta infraestrutura é muito similar a que foi apresentada na seção 2.4, possuindo o mesmo objetivo, mas adaptando para as necessidades nacionais.

2.13. DOC-ICP-15

De acordo com [ITI 2015], o DOC-ICP-15 é a resolução vigente relativa a assinaturas digitais no Brasil. É o documento que registra formalmente as assinaturas digitais em contexto nacional, quais entidades estão envolvidas no processo de assinatura, o ciclo de vida das assinaturas digitais, juntamente com os seus padrões, políticas e perfis acordados. A seguir serão resumidos os principais conceitos usados pelo ITI na ICP-Brasil de acordo com o documento.

2.14. Diferença entre assinatura digital e assinatura eletrônica

De acordo com o [ITI 2015], assinaturas eletrônicas são um conjunto de dados anexado a outro conjunto de dados eletrônico, cujo o objetivo é conferir autenticidade e autoria. Assinatura digital nada mais é do que um tipo de assinatura eletrônica, que usa um par de chaves: uma privada usada para assinar e uma pública para verificar a assinatura.

2.15. Ciclo de vida de uma assinatura digital

Assinaturas digitais devem ser geradas de maneira prática, segura e eficiente, e o fato de que possuem validade limitada, implica que sejam administradas para que funcionem adequadamente até o final de seus prazos de validade e nada além disso, por estas razões, o [ITI 2015] descreve o ciclo de vida de uma assinatura digital na ICP-Brasil, como sendo separado em quatro fases:

- Criação: onde a assinatura é criada, sendo um código que associa a chave privada do signatário ao documento digital.
- Validação: Verificação da assinatura, que alega se ela está válida em relação ao documento eletrônico a que está associada.

- Armazenamento: Compete os cuidados para guardar as assinaturas. E também atualização para mídias de armazenamento mais atuais.
- Revalidação: Realizada para extensão do prazo de validade de uma assinatura. Ou ainda, atualização de segurança para um padrão mais moderno.

Quanto menor o arquivo a ser assinado, menor será o seu resumo criptográfico (*hash*), e consequentemente também sua assinatura digital. E quanto menor o tamanho, mais rápida será sua verificação. Um arquivo pode ser assinado completamente, do início ao fim, ou apenas trechos do arquivo podem ser assinados, como por exemplo um contrato em que várias pessoas necessitam assinar.

Um documento para ser assinado, será primeiramente aplicado em uma função de resumo criptográfico, para a geração do *hash* do documento. Com o *hash* em mãos, este será aplicado juntamente da chave privada do signatário, para gerar a assinatura digital que será anexada ao documento que gerou o *hash*. O processo de criação de uma assinatura é demonstrado conforme este diagrama do [ITI 2015]:

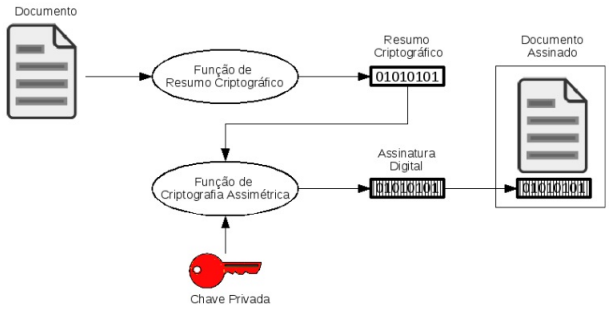

Figura 4. Processo de criação de assinatura digital de acordo com o DOC-ICP-15

Para a verificação da assinatura de um documento, é realizada uma operação de criptografia assimétrica entre a assinatura digital do documento e a chave pública oferecida pelo certificado digital correspondente ao signatário. Com o resultado da operação criptográfica em mãos, este é comparado ao *hash* do documento, se ambos são iguais isto significa que de fato o documento foi assinado pelo portador da chave privada. A seguir, o processo de verificação de uma assinatura é ilustrado pela imagem do [ITI 2015]:

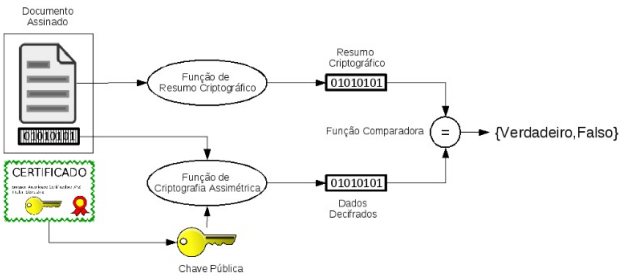

Figura 5. Processo de verificação de assinatura digital de acordo com o DOC-ICP-15

2.16. Padrões de assinatura

As assinaturas digitais dentro da ICP-Brasil, de acordo com o [ITI 2015] possuem padrões diferentes de assinatura conforme o formato do arquivo que está sendo assinado. Os ar-

quivos podem estar no padrão *Cryptographic Message Syntax* (CMS), ou em formatos específicos de *Extensible Markup Language* (XML) ou *Portable Document Format* (PDF).

Desta maneira, em conformidade com o [ITI 2015], três formatos de assinaturas são utilizados na ICP-Brasil:

- Assinatura eletrônica sobre CMS.
- Assinatura eletrônica sobre *Extensible Markup Language Digital Signature* (XML-Sig).
- Assinatura eletrônica sobre PDF.

O padrão CMS é uma estrutura para armazenamento de dados digitais assinados digitalmente de diversas maneiras. Ele possui duas representações de assinaturas assinadas no arquivo, assinatura anexa ou separada, sendo o conteúdo digital incluído na estrutura CMS caso esta esteja anexa. Deve se observar que o documento é assinado na íntegra neste padrão, não podendo ser assinado em trechos exclusivos.

CMS *Advanced Electronic Signature* (CAdES) é uma extensão do CMS, para que assinaturas digitais possam ter estruturação para validação em longo prazo. Consequentemente, com esta extensão, foram criados diferentes formatos de assinaturas, que possuem atributos obrigatórios e não obrigatórios de acordo com a necessidade de cada aplicação. No contexto da ICP-Brasil, um documento que use CAdES precisa estar de acordo com as políticas de assinatura estabelecidas no DOC-ICP 15 para que seja validado.

Arquivos XML, devido a sua estrutura extensível, possuem um padrão específico de assinatura digital, o XML-Sig, o qual permite gerar uma assinatura em apenas uma parte do documento. Possuindo três diferentes maneiras de assinatura digital, sendo ela separada, anexa ou inclusa no próprio conteúdo digital do documento que está sendo assinado.

O padrão XML *Advanced Electronic Signature* (XAdES) é uma extensão do XML-Sig, que de maneira análoga ao CAdES, permite a padronização de assinaturas para validação em longo prazo. Para isso, ele demanda a incorporação de dados adicionais, que implicam na criação de diferentes formas de assinatura. No contexto da ICP-Brasil, um documento que use XAdES precisa estar de acordo com as políticas de assinatura estabelecidas no DOC-ICP 15 para que seja validado.

Os arquivos PDF, são codificados para que caso sejam impressos, a impressão seja exatamente o que está demonstrado no meio eletrônico, independente de variáveis externas, como por exemplo sistema operacional. Tais arquivos possuem um formato específico, PDF Advanced Electronic Signature (PAdES), de assinatura eletrônica, para suportar validação de longe prazo. PAdES sempre será utilizado em documentos PDF exclusivamente, e é anexo na da estrutura do PDF um CMS com conteúdo assinado, sendo todos os *bytes* do arquivo PDF, salvo o bloco do próprio CMS. No âmbito da ICP-Brasil, para que um arquivo PDF com assinatura padrão PAdES seja validada ela precisa estar de acordo com uma das respectivas políticas de assinatura definidas no DOC-ICP 15. As assinaturas deste formato, são visíveis quando o usuário está lendo o PDF.

2.17. Perfis de assinatura

Conforme documentado pelo [ITI 2015], os padrões CAdES, XAdES e PAdES fornecem para a ICP-Brasil muita diversidade de propriedades para incorporar assinaturas digitais

para os mais diversos fins. Devido a esta gama de possibilidades, consequentemente desenvolvedores escolhem apenas um trecho de todos os atributos disponíveis para implementar em seus sistemas, e assim inconsistências podem ocorrer. Para remediar este problema, subconjuntos de atributos específicos usados por certas parcelas de usuários precisam ser identificados, tais subconjuntos são chamados de perfis de assinatura.

Para a ICP-Brasil foi definido um perfil padrão de assinatura, que envolve o CAdES, XAdES, e PAdES. Que é usado para as assinaturas digitais em geral. CAdES usado para arquivos de conjuntos de dados assinados quaisquer, XAdES para arquivos formato XML, e PAdES para arquivos formato PDF.

2.18. Políticas de assinatura

As políticas de assinatura são detalhadas no DOC-ICP-15.03 pelo [ITI 2016], são um conjunto de regras formais para os processos de criação e verificação de assinaturas digitais em âmbito nacional, e definem as bases para validez de uma assinatura. Quando um signatário vai assinar um documento, ele deve escolher uma das políticas disponibilizadas que atendam a sua necessidade, tal assinatura deverá posteriormente ser validada exclusivamente pela mesma política de assinatura.

São 14 políticas de assinatura, criadas para atender uma gama de perfis de usuários diferentes, que usam os padrões CAdES, XAdES e PAdES para assinar e verificar as assinaturas.

3. Verificador de Conformidade de Assinatura Digital

Este capítulo apresenta as motivações e a proposta deste trabalho de conclusão de curso, assim como o detalhamento da estrutura do verificador de assinaturas.

3.1. Introdução

O tema deste trabalho tem como base o verificador de assinaturas digitais criado para o ITI, pelo LabSec. Ele verifica se um dado arquivo encontra-se em conformidade com as normas brasileiras de assinaturas digitais definidas no DOC-ICP-15. O verificador é disponibilizado em um *website*[2] para os usuários, e foi modelado para ser acessado a partir de um *web browser*.

3.2. Proposta

A proposta deste trabalho é a melhoria da acessibilidade deste serviço de verificação de assinaturas digitais, por meio da implementação de um aplicativo móvel para as multiplataforma.

Tendo em mente que as diferentes plataformas alvo do aplicativo, é notório o problema de que existem muitas particularidades em cada plataforma, e isto implica na implementação de um código exclusivo para atender a cada uma das plataformas de maneira específica. Mas como o modelo da aplicação em si seria basicamente o mesmo, então foi procurada uma solução em que fosse possível manter um modelo geral para todas as plataformas, mas que também pudesse atender as particularidades de cada plataforma, e a ferramenta encontrada para remediar este problema foi o Apache Cordova.

[2]https://verificador.iti.gov.br/

O Apache Cordova permite que o programador crie um aplicativo, a partir de uma única base de código, e compile o programa para diversas plataformas, resolvendo as necessidades de cada plataforma.

3.3. Verificador de Conformidade

Primeiramente foi discutida a maneira de como seria a melhor maneira de modelar a aplicação. Dois possíveis caminhos de implementação foram cogitados, cada um com suas vantagens e desvantagens.

O primeiro modelo seria um aplicativo independente, em que ele possuísse dentro de seu pacote, todos os recursos necessários para seu funcionamento. Permitindo que o próprio dispositivo do usuário, independentemente de acesso à *internet*, pudesse realizar a verificação de assinaturas. O segundo modelo seria uma aplicação *full-stack*, em que o usuário enviasse o arquivo a ser verificado para um servidor externo que realizaria a verificação e retornasse o resultado. A aplicação ficaria dividida entre o *front-end*, interface de acesso do usuário, e o *back-end* (adaptado pelo LabSec, à partir do existente verificador do *site* do ITI), servidor com um *web service* verificador das assinaturas.

A vantagem principal do primeiro modelo é que o usuário, ao contrário do segundo modelo, pode verificar suas assinaturas a qualquer momento, mas conveniência possui um preço: desempenho. O aplicativo teria que conter toda a biblioteca verificadora, o que seria altamente ineficiente devido ao tamanho que o arquivo final teria, além do próprio dispositivo do usuário ter que processar o processo de verificação. Já no segundo modelo, a vantagem principal é de que o dispositivo do usuário ficaria com uma aplicação leve e compacta, que se limitaria apenas a requisitar a computação da verificação para um servidor(implementado pelo LabSec). Em relação ao usuário, a única desvantagem seria a necessidade de conexão de *internet* com velocidade suficiente para enviar o arquivo e receber a resposta do servidor. Outra desvantagem é a necessidade de hospedar o *web service* do verificador.

Dois problemas foram levantados para serem enfrentados durante a implementação do aplicativo: o problema de fazer o sistema operacional alvo utilizar a biblioteca verificadora escrita em Java, e o problema de adaptar a interface em cada uma das plataformas. Estes problemas foram levados em consideração quando foi acordado que o segundo modelo seria mais adequado, considerando que a usabilidade do usuário é a maior prioridade, e a dificuldade de adaptar toda a biblioteca em cada contexto de plataformas diferentes. Foi então que o LabSEC criou um *web service* hospedando a biblioteca verificadora necessária para a execução da tarefa, e o disponibilizou[3] para acesso em *browsers*, e desta maneira, foi decidido que o aplicativo usaria o mesmo *web service* deste *site*.

Sobre o *back-end* que fornece o serviço de verificação, foi iniciado como um trabalho de conclusão de curso realizado por [de Oliveira 2012] e o LabSEC deu continuidade nesta implementação. Basicamente ele recebe uma requisição via POST com o arquivo a ser verificado, o arquivo passa pelo processo de verificação na biblioteca e resulta em um XML que é depois convertido em JSON(para o *webservice* usado neste trabalho, ou HTML(para o *site* do ITI, que é retornado para a origem da requisição e apresentado para o usuário.

[3]https://pbad.labsec.ufsc.br/homologacao/

3.4. Funcionamento do verificador

A tarefa proposta a ser realizada pela aplicação é: O aplicativo deve receber arquivos fornecidos pelo usuário, via interface, de maneira acessível e simplificada. Em seguida enviar o arquivo para o *webservice*, o qual realizará o processamento do arquivo enviado e retornará um arquivo JSON. Com o resultado do processamento em mãos basta o aplicativo apresentar para o usuário em sua interface os resultados.

Foram criados os seguintes casos de uso para o aplicativo:

- Caso 1: Verificar se um arquivo é assinado.
- Ator: Usuário do aplicativo
- Fluxo básico: O usuário quer saber se o arquivo possui uma assinatura em sua estrutura. Para isso ele abre o aplicativo em seu celular, em seguida clica no botão ”verificar”, o qual pede para que seja selecionado um arquivo no sistema de arquivos. Após a seleção do usuário o arquivo é enviado e uma resposta dizendo se o arquivo é ou não assinado é mostrada na tela do celular.

É importante salientar que o arquivo que o usuário selecionar pode ou não estar assinado, consequentemente o programa terá de tratar tais eventos. Além disso, o usuário precisa estar conectado com a *internet*, para que o arquivo possa ser enviado ao *webservice*.

- Caso 2: Verificar a validade de uma assinatura.
- Ator: Usuário do aplicativo
- Fluxo básico: O usuário quer verificar se um arquivo assinado em seu dispositivo está em conformidade com as normas da ICP-Brasil, para isso ele abre o aplicativo em seu celular, em seguida clica no botão verificar, o qual pede para que seja selecionado um arquivo no sistema de arquivos. Após a seleção do usuário o arquivo é enviado e uma resposta apresenta o relatório da verificação na tela do celular, dizendo se o arquivo possui uma assinatura com validade e seus detalhes.

A seguinte figura ilustra os casos de uso:

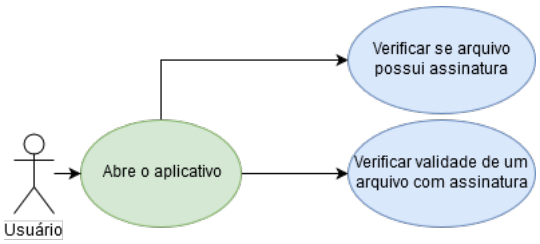

Figura 6. Casos de uso da aplicação.

4. Ferramentas e Tecnologias

Neste capítulo serão detalhadas as ferramentas e conceitos usados no desenvolvimento do aplicativo proposto para este trabalho.

4.1. *Web Services*

Um *web service* genericamente falando, é um serviço oferecido por um dispositivo eletrônico conectado em uma rede para outros dispositivos eletrônicos que possuem conexão. Mais especificamente falando, é um servidor que está executando um programa que oferece uma solução para uma dada entrada ou requisição externa. São oferecidos por diversas empresas como soluções para *web* e estão relacionados a basicamente qualquer serviço da *internet* e aplicativos que necessitem de informações da *web*, podendo também ser modelados de forma específica de acordo com um padrão de uma certa empresa ou marca.

Na prática, um *web service* providencia uma interface para um serviço que resolve um problema, hospedado num servidor. Um programa de *front-end* pode requisitar a execução de uma função externa à ele, como por exemplo uma computação que o dispositivo do usuário não teria capacidade de realizar, e retorna um conjunto de dados, normalmente em formato XML ou JSON, que será tratado no *front-end* da aplicação para mostrar o resultado para o usuário.

4.2. Apache Cordova

Cordova, originalmente chamado de *PhoneGap*, feito pela [Apache 2018] é um *framework* de desenvolvimento de aplicações móveis. Ele permite a construção de aplicações para dispositivos móveis usando CSS3, HTML5, e JavaScript ao invés de utilizar API's específicas de plataformas como as disponíveis para *Android*, *iOS* e *Windows Phone*.

Ele estende as características do HTML e JavaScript para trabalhar com o dispositivo, unificando tudo em uma única base de código, que pode ser compilado para diversas plataformas. As aplicações resultantes são híbridas, ou seja, não são nem aplicações móveis nativas, nem puramente baseadas em *web*.

Tecnicamente falando, a interface de uma aplicação Cordova é uma *WebView* que ocupa completamente a tela do dispositivo, e roda em seu *container* nativo. A seguinte imagem ilustra o funcionamento básico do Cordova:

4.3. Kits de Desenvolvimento de Web Apps

4.4. Ionic

Criado pela [Drifty 2015], Ionic é um kit de desenvolvimento de código aberto para aplicativos móveis híbridos. Ele foi originalmente construído sobre AngularJS (mantido pelo Google) e Apache Cordova. Atualmente permite que o usuário utilize qualquer *framework* de interface de usuário. Também possui uma biblioteca de componentes de interface que facilitam a integração multiplataforma de aplicações web híbridas, usando Apache Cordova. Um projeto Ionic, quando compilado, gerará um código específico para cada plataforma alvo a partir de um código base compartilhado. Depois com os códigos específicos de cada plataforma, o desenvolvedor pode realizar adições de elementos exclusivos de cada plataforma em seus respectivos ambientes de desenvolvimento caso haja necessidade, antes de realizar a compilação final que gerará o aplicativo em si.

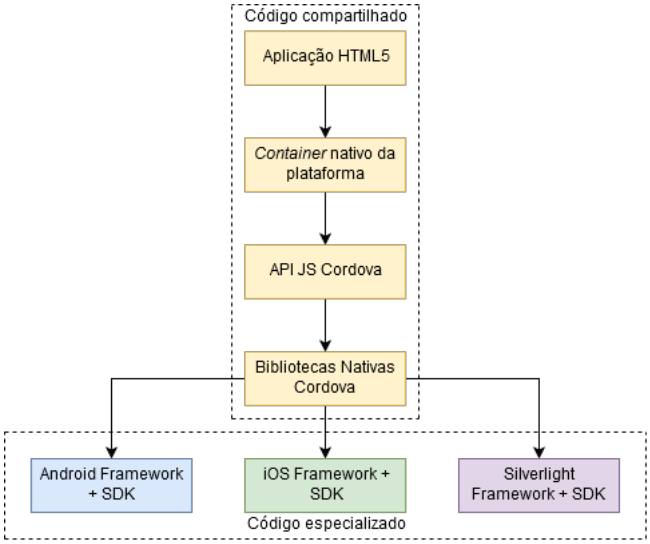

Figura 7. Fluxo básico de uma aplicação Cordova.

4.5. Android Studio

O Android Studio do [Google 2019], é o ambiente de desenvolvimento integrado oficial para o desenvolvimento de aplicativos para o sistema operacional Android. Ele possui diversas recursos para auxiliar o programador, tais como suporte a Gradle, ferramentas específicas de refatoração de código para Android, um emulador virtual para testar a sua aplicação, entre muitos outros recursos. Ele suporta linguagens de programação populares, como Java, C++ e Go. Também permite compilação da aplicação para diversas versões de Android e alerta ao programador sobre a compatibilidade de sua aplicação relativa a possíveis diferenças de versões de Android. No âmbito deste trabalho será usado o Android Studio para compilar o código específico para a plataforma Android gerado pelo Ionic.

4.6. XCode

XCode, feito pela [Apple 2019], é o ambiente de desenvolvimento oficial de aplicativos para a plataforma iOS. Ele suporta diversas linguagens de programação populares, como C, C++, Java, Python, entre muitas outras. Também possui a capacidade de gerar binários que contém código para múltiplas arquiteturas, permitindo que o software rode em processadores PowerPC e Intel x86 em 32 e 64 bits de arquitetura. No âmbito deste trabalho será usado o XCode para realizar a compilação do código específico de plataforma iOS gerado pelo Ionic.

5. Implementação

Este capítulo tem como o objetivo demonstrar e documentar como foi o processo de implementação e suas respectivas decisões de projeto e design.

5.1. Implementação no Ionic Framework

A pesquisa por um *framework* para utilizar o Apache Cordova para gerar código híbrido, resultou na utilização do Ionic Framework para realização do desenvolvimento do *front-end*. As seguintes subseções detalham especificamente cada parte do desenvolvimento.

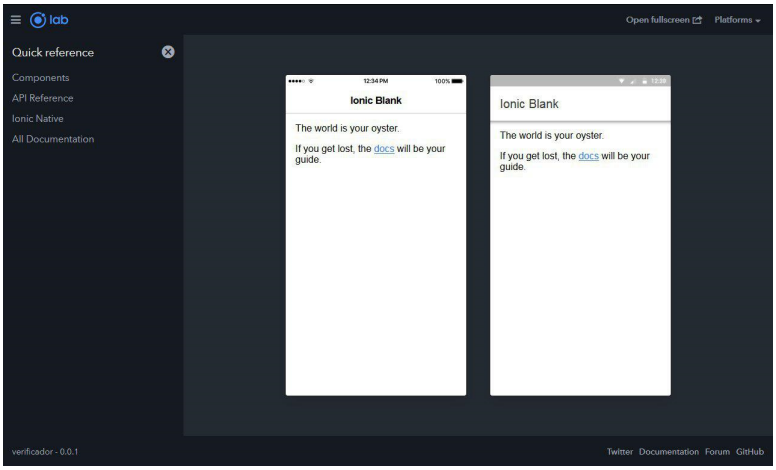

Figura 8. Projeto iniciado no Ionic.

5.2. Criação do projeto

Após a instalação do Ionic e suas dependências, o processo de implementação se deu início, com um estudo sobre a estrutura do *framework*, resultando na exploração de suas funcionalidades e entendimento de suas possíveis limitações.

O primeiro passo foi a criação de um projeto em branco, para estudar os conceitos básicos do *framework*, e posteriormente ampliar o mesmo para atender todas as funcionalidades planejadas. Para gerar o projeto inicial, foi executado o seguinte comando: **ionic start ¡name¿ ¡template¿ [options]**.

Onde *¡name¿* é o nome do seu projeto, *¡template¿* é um parâmetro opcional que lhe permite escolher um modelo pré-modelado e *[options]* são parâmetros adicionais opcionais. Para este projeto foi gerado um projeto em branco, demonstrado a seguir:

5.3. Estrutura do projeto

Um projeto Ionic quando criado gera uma estrutura de pastas que organizam as bibliotecas, módulos e partes da aplicação. A estrutura do projeto ”verificador”é apresentada a seguir:

- O projeto em si se encontra dentro da pasta *top level* ”verificador”. Dentro desta pasta também se encontra o arquivo *config.xml*, responsável pelos parâmetros de configuração do projeto.
- A pasta ”e2e”contém o *Protractor*,uma biblioteca de Angular para escrita de suíte de testes para a aplicação, não foi usado neste trabalho, pois não foram criados testes *end to end*.
- A pasta ”node modules”contém os pacotes NodeJs do projeto.
- A pasta ”platforms”contém o código gerado específico das plataformas alvo requisitadas.
- A pasta ”plugins”contém os plugins Cordova que a aplicação usa.
- A pasta ”resources”contém ícones e elementos de interface específicos de cada plataforma alvo.
- A pasta ”src”contém o código não compilado completo da aplicação. E a pasta ”www”contém o código compilado executável em *browser*.

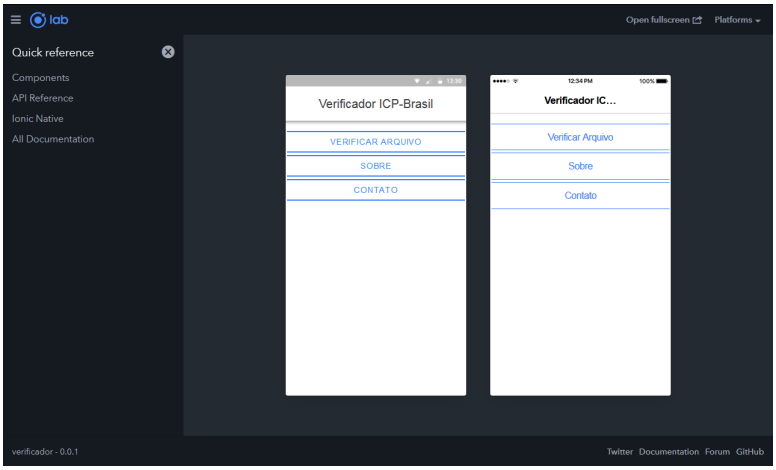

Figura 9. Primeira parte feita no Ionic

5.4. Criação da interface

Após o projeto estar iniciado o próximo passo foi implementar os botões para que o usuário possa realizar as tarefas. Para isto é possível criar os arquivos respectivos das páginas que nossa aplicação irá usar, ou pode-se usar o comando de geração de *templates* de código de certos componentes pré-prontos: **ionic generate ¡type¿ ¡name¿ [options]**.

Onde *¡type¿* é o tipo de modelo desejado e *¡name¿* é o nome que deseja para o componente a ser adicionado e *[options]* são parâmetros adicionais opcionais. Tais *templates* são criados dentro de uma subpasta com seu nome dentro da pasta *src* do projeto. No caso, foi usado o comando de gerar páginas, o que gera uma subpasta que contém um arquivo Typescript para a lógica da página, um arquivo de estilos *.scss* e um arquivo HTML para a apresentação da página, além de alguns outros arquivos de configuração. Consequentemente, a seguinte interface foi implementada:

Foi adicionado o botão ”Verificar Arquivo”, o qual realizará a tarefa principal da aplicação. Também foram adicionados um botão ”Sobre”e um botão ”Contato”, para que o usuário possa obter mais informações a respeito da aplicação. Ao clicar em qualquer um destes botões, o usuário é direcionado para a respectiva página por meio de uma referência HTML, que no caso do botão verificar, também chama um *script* que invoca a função *verificarArquivo()*.

5.5. Implementação das funções

A estrutura básica da lógica da aplicação é apresentada na imagem a seguir:

Primeiro o usuário ao clicar no botão verificar, isto irá disparar a função *verificarArquivo()* encontrada na página *report*, tal função primeiro irá detectar a respectiva plataforma do dispositivo que está executando, para encaminhar para o devido *file picker* específico para sua plataforma, visto que existe um específico para Android e um específico para iOS. Quando detectada a plataforma, ele irá direcionar para a execução do método *pickFile()* específico para esta plataforma, quando executado realizará uma requisição na interface do usuário para o mesmo escolher o arquivo que deseja enviar para verificar. Após selecionar o arquivo, o método gera como resposta uma *string* com o diretório do arquivo selecionado.

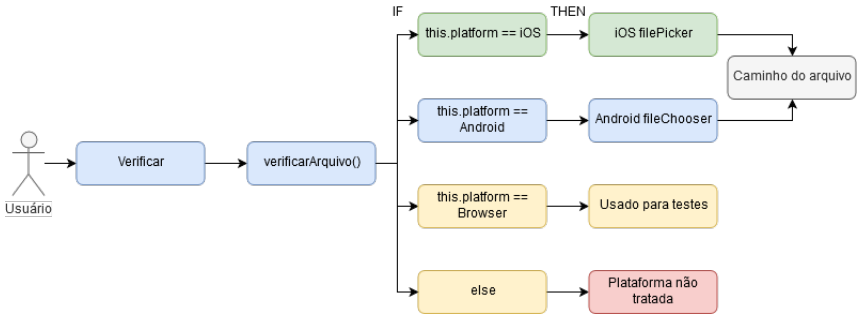

Figura 10. Fluxo da requisição

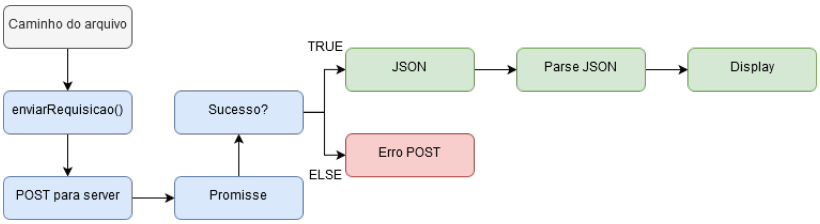

Figura 11. Fluxo do POST para *back-end*

Com isso, é invocado o método *enviarArquivo(string uri)* que recebe o diretório como parâmetro, para realizar a chamada assíncrona que envia o arquivo para o servidor. Para realizar tal requisição é usado o *plugin* HTTP disponibilizado pelo Cordova, e que foi adaptado pelos desenvolvedores do Ionic, para realizar uma requisição do tipo POST para o endereço do *back-end*, enviando anexa a requisição o arquivo, gerando uma *promisse* de resposta. Se a *promisse* for cumprida, um JSON será recebido, caso contrário um alerta de erro é emitido. Com o JSON do relatório em mãos, o mesmo é enviado para a página do relatório e é tratado para ser apresentado no HTML da página.

5.6. Geração de código para plataformas específicas

Para gerar o código específico para uma plataforma, de dentro do diretório do projeto basta digitar no terminal o seguinte comando: **ionic cordova prepare ¡platform¿**.

Onde *"platform"* é a plataforma alvo de geração de código desejada, neste projeto foram compiladas as versões específicas de *iOS* e *Android* com os comandos: **ionic cordova prepare android; ionic cordova prepare ios**.

Cada comando gera uma pasta respectiva à sua plataforma dentro do diretório *"platforms"* da aplicação, que contém todas as plataformas compiladas. O código específico gerado pode ser posteriormente compilado no *Android Studio* e no *XCode*, isto irá gerar a aplicação final que pode ser emulada no computador para testes ou instalada no celular do usuário. A seguir um exemplo de compilação realizada no *Android Studio*:

É importante salientar o fato de que neste ponto, modificações adicionais podem ser adicionadas no projeto *Android* (assim como poderia ser feito no *XCode* com *iOS*), caso fossem necessárias funcionalidades que apenas pudessem ser implementadas usando recursos particulares de cada plataforma, e que não fossem suportadas pelo *Cordova*.

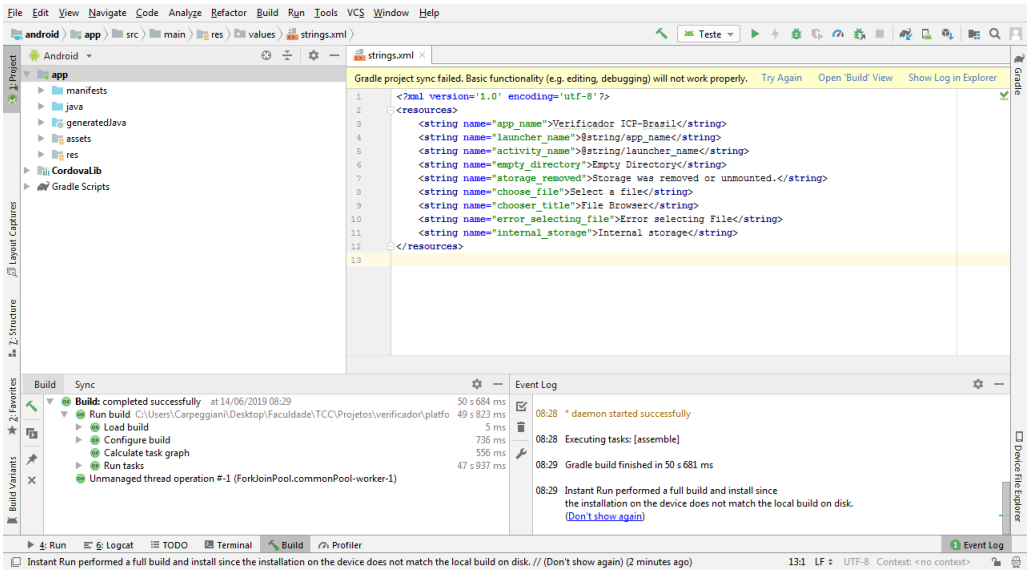

Figura 12. Captura de tela durante compilação no *Android Studio*

5.7. Teste da aplicação final

Com o objetivo de verificar o desempenho do aplicativo, foram realizadas diversas requisições de verificação, e medidos seus tempos de resposta, para estimar uma média de quanto tempo um determinado tipo de arquivo leva para ser processado pelo serviço do verificador.

A aplicação foi testada com compilação alvo para *Android* versão 8. Foi utilizado o programa [Inc. 2019], para medir o tempo de resposta das requisições. O tamanho da amostra é de 20 requisições, para cada um dos três tipos de arquivo de assinatura, com os mesmos arquivos, para manter um tamanho constante. O tamanho dos arquivos testados é de 203 KB para PDF, 33KB para XML e 3KB para P7S.

A seguintes figuras apresentam os dados: Figura 13 e 14.

O tempo médio de resposta de requisição calculado foi de 302,4 milissegundos para P7S, 325 milissegundos para XML e 596,3 milissegundos para PDF.

5.8. Conclusão

Esta documentação, poderá ser usada posteriormente para beneficiar futuros trabalhos relacionados. Fica então demonstrado o básico do uso do Ionic *framework*, introduzindo o leitor a estrutura do projeto implementado neste trabalho, com seus detalhes particulares de implementação, e resultado final obtido. Facilitando assim a implementação de futuros trabalhos.

6. Conclusão

Com base no que foi apresentado, conclui-se que o objetivo de implementação de uma aplicação multiplataforma, com uma única base de código comum, que foi compilada para as plataformas *Android* e *iOS*, que permite o usuário ter o conforto e conveniência de verificar assinaturas de acordo com as normas estabelecidas pela atual legislação brasileira onde quer que esteja que possua acesso à *internet*.

Tempo de resposta de requisição			
Amostra	**Delay P7S**	**Delay XML**	**Delay PDF**
1	304	300	641
2	300	320	583
3	295	300	598
4	342	300	583
5	312	300	579
6	300	310	597
7	302	360	584
8	216	300	574
9	342	320	645
10	299	290	676
11	297	310	575
12	308	291	596
13	296	308	595
14	303	301	574
15	296	319	604
16	309	319	600
17	296	613	576
18	299	325	584
19	313	302	588
20	319	312	574
Soma:	6048	6500	11926
Média:	302,4	325	596,3

Figura 13. Tabela de amostragem de tempo de resposta de requisição.

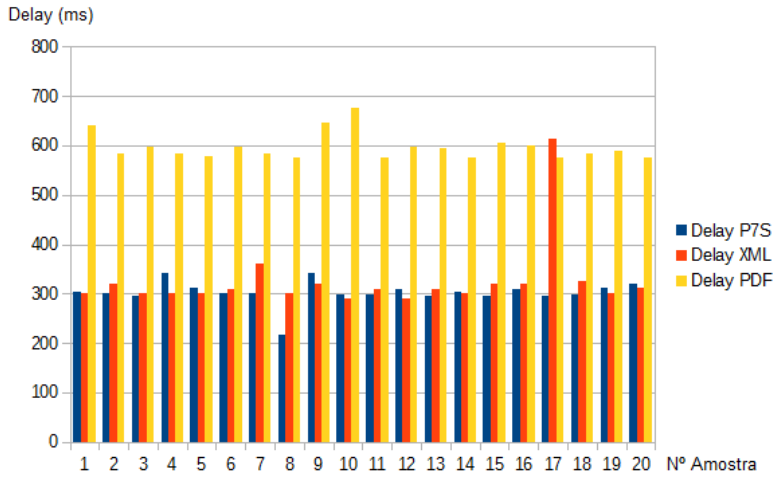

Figura 14. Gráfico da tabela de amostragem.

O objetivo da documentação do processo de desenvolvimento também foi cumprido, tendo sido realizada a apresentação de todo o processo da criação da aplicação, e acompanhado da entrega desta monografia o código fonte devidamente documentado.

Esta aplicação em seu estado atual, gera um impacto positivo, visto que as assinaturas digitais estão cada vez mais presentes no cotidiano do brasileiro, o aplicativo pode vir a facilitar muitos processos existentes e futuros, como por exemplo, verificação de documentos.

Apesar de cumprir os objetivos estabelecidos para este trabalho, esta não é uma solução perfeita para o problema, o que viabiliza a possibilidade de realização de trabalhos futuros de extensão focados em otimização de funcionamento, e *design*.

Muitos pontos de melhoria existem, como a melhoria da usabilidade do aplicativo para ser mais acessível a todos os tipos de usuários, suporte a deficientes, e uma melhor apresentação do relatório, para que este fique mais fácil de ser interpretado pelo usuário. O suporte a múltiplos arquivos de assinatura simultaneamente ou assinaturas em lote. A melhoria do desempenho de processamento de arquivos de assinatura grandes. E por último, a extensão deste aplicativo para que o mesmo possa também realizar verificação de diplomas com certificação digital por meio de integração com o serviço verificador de diplomas brasileiro.

Referências

Anderson, R. J. (2010). *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd edition. ISBN: 978-0-47006-852-6.

Apache (2018). *Apache Cordova*. https://cordova.apache.org/.

Apple (2019). *XCode*. https://developer.apple.com/xcode/.

Choudhury, S. (2002). *Public Key Infrastructure Implementation and Design*, 1st edition. ISBN: 978-0-76454-879-6.

de Oliveira, M. S. (2012). *Modelagem de um Software Orientado à Componentes para Assinatura Digital*. UFSC.

Drifty (2015). *Ionic Framework*. https://ionicframework.com/.

Exame, R. (2018). *Transformação digital aumenta oportunidades para as empresas*. https://exame.abril.com.br/tecnologia/transformacao-digital-aumenta-oportunidades-para-as-empresas/.

Google (2019). *Android Studio*. https://developer.android.com/studio.

Higa, P. (2018). *Celular se torna principal meio de acesso à internet no Brasil*. https://tecnoblog.net/252838/celular-principal-meio-acesso-a-internet-brasil-tic-domicilios-2017/.

Inc., P. (2019). *Postman*. https://www.getpostman.com/.

ITI (2015). *VISÃO GERAL SOBRE ASSINATURAS DIGITAIS NA ICP-BRASIL*. https://www.iti.gov.br/legislacao/documentos-principais.

ITI (2016). *REQUISITOS DAS POLÍTICAS DE ASSINATUR ADIGITAL NA ICP-BRASIL*. https://www.iti.gov.br/legislacao/documentos-principais.

ITI (2017a). *Entes da ICP-Brasil*. http://www.iti.gov.br/icp-brasil/57-icp-brasil/76-como-funciona.

ITI (2017b). *O ITI*. http://www.iti.gov.br/institucional/43-institucional/89-o-iti.

ITI (2018). *Verificador de Conformidade*. https://verificador.iti.gov.br/.

Katz, J. and Lindell, Y. (2007). *Introduction to Modern Cryptography: Principles and Protocols*, 1st edition. ISBN: 978-1-58488-551-1.

Oliveira, D. (2018). *Banco Sumitomo Mitsui Brasileiro aposta em assinatura digital*. https://computerworld.com.br/2018/09/04/banco-sumitomo-mitsui-brasileiro-aposta-em-assinatura-digital/.

Stallings, W. (2005). *Cryptography and Network Security Principles and Practices*, 4th edition. ISBN: 978-0-13187-316-2.

Wakka, W. (2018). *Motoristas do DF terão documento digital do carro (CRLV)*. https://canaltech.com.br/governo/motoristas-do-df-terao-documento-digital-do-carro-crlv-ja-na-segunda-27-121104/.