UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS TRINDADE — CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO CIÊNCIA DA COMPUTAÇÃO

Guilherme de Moraes Sartori

Software de Código Aberto para Prestadores de Serviço de Confiança

Florianópolis
2022

Guilherme de Moraes Sartori

Software de Código Aberto para Prestadores de Serviço de Confiança

Trabalho Conclusão do Curso de Graduação em Ciências da Computação do Campus de Trindade da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Ciências da Computação
Orientador: Prof. Jean Everson Martina
Coorientador: Pablo Rinco Montezano

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Sartori, Guilherme de Moraes
Software de Código Aberto para Prestadores de Serviço de Confiança / Guilherme de Moraes Sartori ; orientador, Jean Everson Martina, coorientador, Pablo Rinco Montezano, 2022.
78 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2022.

Inclui referências.

1. Ciências da Computação. 2. Segurança. 3. Criptografia de Chave Pública. 4. Prestador de Serviço de Confiança. 5. Infraestrutura de Chaves Públicas. I. Martina, Jean Everson. II. Montezano, Pablo Rinco. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Titulo.

Guilherme de Moraes Sartori

Software de Código Aberto para Prestadores de Serviço de Confiança

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Ciência da Computação" e aprovado em sua forma final pelo Curso de Ciência da Computação.

Florianópolis, 6 de Agosto de 2022.

_____

Prof. Jean Everson Martina
Coordenador do Curso

Banca Examinadora:

_____

Jean Everson Martina
Orientador
Universidade Federal de Santa Catarina

_____

Pablo Rinco Montezano
Coorientador
Universidade Federal de Santa Catarina

_____

Lucas Mayr de Athayde
Avaliador
Universidade Federal de Santa Catarina

RESUMO

Um Prestador de Serviço de Confiança (PSC) é uma entidade responsável por prover serviços de gerenciamento de chaves privadas e geração de assinaturas digitais. PSCs são auditados e fiscalizados no Brasil pelo Instituto Nacional de Tecnologia da Informação (ITI) e devem funcionar dentro dos padrões estabelecidos para a Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil). A ICP-Brasil emite certificados digitais que garantem, através de uma cadeia de confiança, que uma certa chave pública, e sua correspondente chave privada, pertencem ao usuário final que recebeu o certificado. A chave privada associada ao certificado pode ser utilizada para a geração de assinaturas digitais, que possuem características e aplicações semelhantes a assinaturas convencionais. Para que essas características sejam mantidas, a chave privada devem ser armazenadas de forma que apenas o proprietário do certificado possua acesso a ela. No entanto, o gerenciamento de chaves privadas e de assinaturas digitais pode ser um processo bastante inconveniente. PSCs atuam para facilitar esse gerenciamento para os usuários finais, e fazem isso seguindo rígidos requisitos de segurança para manter propriedades de segurança essenciais. Apesar de serem uma importante parte da ICP-Brasil, não existem software aberto que oferece as funcionalidades necessárias para o funcionamento de um PSC. Este trabalho trata do desenvolvimento de um sistema web de código aberto que provê esse serviço. O sistema web foi desenvolvido em Java utilizando o framework Spring e banco de dados MySQL através de prototipação seguida de sucessivos melhoramentos. No corpo do trabalho, a API do sistema web e seu funcionamento será detalhado.

Palavras-chave: Segurança. Criptografia de Chave Pública. Provedor de Serviço de Confiança.

ABSTRACT

A Trust Service Provider (TSP) is an entity responsible for providing services of private key management and generation of digital signatures. TSPs are audited and inspected in Brazil by the Instituto Nacional de Tecnologia da Informação (ITI) and must work under the standards established for the Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil). ICP-Brasil issues digital certificates, that assure, via a trust chain, that a certain public key, and its corresponding private key, belong to the end user that received the certificate. The private key associated to the certificate can be used to generate digital signatures, which possess characteristics and applications similar to those of conventional signatures. In order for those characteristics to be maintained, the private key must be stored so that only the owner of the key pair has access to it. However, the management of private keys and digital signatures can be a very inconvenient process. TSPs act to facilitate this management for end users, following strict security requirements to keep essential security properties. Despite being an important part of ICP-Brasil, there is no open source software that offers all of the required features for the operation of a TSP. This paper deals with the development of an open source web system that provides this service. The system was developed in Java using the Spring framework and MySQL database through prototyping followed by successive upgrading. In the body of this paper, the API of the web system and details of its behavior will be detailed.

Keywords: Security. Public key cryptography. Trust service provider.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AC Autoridade Certificadora

AES Advanced Encryption Standard

API Application Programming Interface

HSM Hardware Security Module

ICP-Brasil Infraestrutura de Chaves Públicas Brasileira

ITI Instituto Nacional de Tecnologia da Informação

JSON JavaScript Object Notation

JWT JSON Web Token

KMIP Key Management Interoperability Protocol

LPSC Lista de Prestadores de Serviço de Confiança

PSC Prestador de Serviço de Confiança

REST Representational state transfer

SGBD Sistema de Gerenciamento de Banco de Dados

SUMÁRIO

1 INTRODUÇÃO

A criptografia de chave pública, ou criptografia assimétrica, é um esquema de criptografia que envolve um par de chaves, uma chave pública e uma chave privada. Cada chave pública é distribuída e a sua correspondente chave privada é mantida secreta. Dados que são encriptados com a chave pública somente podem ser decriptados com a chave privada e vice-versa (IBM, 2022).

Para a comunicação utilizando esse esquema de criptografia, o mensageiro encripta a mensagem com a chave pública do destinatário, porque dessa forma apenas o destinatário será capaz de decriptar a mensagem. Realizando o processo com a chave privada, é criado o que é conhecido como uma assinatura digital (IBM, 2022).

A assinatura digital é diferente da cifragem com chave pública para comunicação encriptada. A assinatura digital é um dado que só pode ser gerado pela chave privada, mas que pode ser verificada por qualquer um através da correspondente chave pública. Isso faz com que, assumindo que a segurança da chave privada não foi violada, garantidamente a assinatura digital foi gerada pelo proprietário do par de chaves.

Infaestruture de Chaves Públicas, ou *Public Key Infrastructure* (PKI), é o conjunto de políticas, processos e papéis utilizados para gerenciamento de chaves públicas através de certificados digitais. PKI surgiu na década de 1990 para ajudar na governância de chaves criptográficas através da emissão e do gerenciamento de certificados digitais. Esses certificados são utilizados para verificar a posse de um certo par de chaves (KEYFACTOR, 2022).

Certificados digitais são emitidos e digitalmente assinados por Autoridades Certificadoras (AC). Essas ACs por sua vez tem seus certificados emitidos por outras ACs, criando uma cadeia de certificação que inicia em uma AC raiz com um certificado autoassinado (KEYFACTOR, 2022). Se uma entidade confia na AC raiz, então ela também confiará nos certificados de ACs que tenham sido emitidos pela AC raiz e também em certificados emitidos por essas ACs.

A Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil) mantém uma cadeia hierárquica de confiança para emissão e verificação de certificados digitais (ITI, 2021). As ACs raiz brasileiras assinam e emitem certificados para ACs intermediárias, que assinam e emitem certificados para ACs finais, que, finalmente assinam e emitem certificados para usuários finais.

De posse de um par de chaves, um usuário final pode obter um certificado a partir de uma AC final e realizar assinaturas que poderão ser verificadas através da chave pública de seu certificado, que tem sua validade assegurada pela assinatura da AC final e pelo resto da cadeia de certificação brasileira.

Além dos certificados para verificar a posse de um par de chaves, também é necessária a segurança do próprio par de chaves, particularmente da chave privada. Prestadores de Serviço de Confiança (PSCs) provêm esse serviço.

PSCs, dentro do contexto da ICP-Brasil, provêm armazenamento de chaves privadas para usuários finais, geração e verificação de assinaturas. PSCs são auditados e fiscalizados pelo Instituto Nacional de Tecnologia da Informação (ITI) e devem atendar às normativas estabelecidos pela ICP-Brasil (ITI, 2019).

O DOC-ICP-17.01 determina os vários requisitos de segurança de PSCs que operam na ICP-Brasil, incluindo especificações de segurança pessoal, segurança física, segurança lógica, requisitos de armazenamento de chaves, entre outros. Esses são requisitos chave para garantir o pleno funcionamento, a segurança e a capacidade de auditoria de PSCs.

O ITI mantém a Lista de Prestadores de Serviço de Confiança da ICP-Brasil. A LPSC (ITI, 2021) registra os PSCs credenciados para oferecer serviços à ICP Brasil e apresenta no presente momento sete entidades credenciadas.

Apesar de ser um componente importante da ICP-Brasil, não existe um software aberto ou livre disponível que realiza as funções necessárias para um PSC. Um sistema desse tipo poderia facilitar no surgimento de novas PSCs na ICP-Brasil ou poderia até mesmo substituir sistemas existentes.

A contribuição deste trabalho é, portanto, com o desenvolvimento de um software para PSCs de código aberto que siga padrões de segurança razoáveis que possa servir como ponto de partida para um eventual sistema completo que possa ser colocado em produção na ICP-Brasil.

1.1 OBJETIVOS

Nesta seção serão descritos o objetivo geral e os objetivos específicos deste TCC.

1.1.1 Objetivo Geral

O objetivo geral do trabalho é o desenvolvimento de um sistema web de código aberto que implemente funcionalidades básicas de Prestadores de Serviço de Confiança de forma razoavelmente segura. O trabalho busca criar uma base sólida que possa ser expandida para um eventual software completo ou ser utilizada como referência para um futuro software completo.

1.1.2 Objetivos Específicos

- Desenvolvimento e entrega do código fonte de um sistema web para PSCs
- O sistema desenvolvido dever permitir o armazenamento seguro de pares de chaves
- O sistema deve permitir a geração de assinaturas digitais a partir das chaves e a verificação dessas assinaturas digitais
- As chaves privadas no sistema devem apenas ser utilizáveis para gerar assinaturas pelos seus proprietários
- Desenvolvimento e entrega da documentação da API do sistema web desenvolvido

1.2 ESCOPO

O desenvolvimento se limita à implementação de um sistema web que forneça as funcionalidades de armazenamento de pares de chaves assimétricas e assinatura digital. Um sistema de auditoria não faz parte do escopo deste trabalho, mas a implementação do sistema busca ser razoavelmente segura dentro do limite do possível. A implementação do sistema realizada neste trabalho não teve como objetivo seguir as especificações do DOC-ICP-17.01 e, portanto, a versão final do sistema desenvolvido neste trabalho não busca ser um sistema já preparado para ser colocado em produção na ICP-Brasil.

1.3 ESTRUTURA DO TRABALHO

A seção 1 apresenta a motivação do trabalho, seus objetivos e alguns conceitos básicos. A seção 2 elabora sobre os conceitos teóricos considerados como principais para a compreensão do trabalho. A seção 3 fala sobre a análise dos requisitos e problemas envolvidos na implementação do sistema e discorre sobre as tecnologias utilizadas e soluções escolhidas para a implementação do sistema. A seção 4 apresenta a versão final do sistema, seu funcionamento, sua API e suas limitações. A seção 5 apresenta as conclusões e questões para serem elaboradas em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção será apresentado alguns princípios teóricos necessários para entendimento do sistema desenvolvido.

2.1 CRIPTOGRAFIA SIMÉTRICA

Criptografia simétrica, também chamada de criptografia convencional ou criptografia de chave única era o único tipo de criptografia em uso antes do desenvolvimento de criptografia de chave pública na década de 1970 (STALLINGS, 2013). Na criptografia simétrica, uma única chave secreta é utilizada tanto no processo de cifragem como decifração de dados (IBM, 2022).

Um esquema de criptografia simétrica possui cinco fatores: o texto ou dados originais, chamado comumente de *plaintext*; o algoritmo de cifragem; a chave secreta, que é utilizada tanto no algoritmo de cifragem como no de decifração; o *ciphertext* ou texto cifrado, resultado da execução do algoritmo de cifragem sobre o *plaintext*; o algoritmo de decifração, que transforma o texto cifrado de volta em *plaintext* (STALLINGS, 2013). Em muitos casos, o algoritmo para decifrar é o mesmo que o algoritmo para cifrar, apenas executado ao contrário.

Existem dois requisitos para o uso seguro de criptografia simétrica. Primeiro, o algoritmo criptográfico precisa ser poderoso ao ponto de que, no mínimo, um atacante que conhece o algoritmo e tem acesso a múltiplos *ciphertexts* não será capaz de decifrar *ciphertexts* ou descobrir a chave. Segundo, tanto aquele que envia o texto cifrado quanto aquele que o recebe devem ambos terem recebido cópias da chave secreta de uma forma segura e devem mantê-la segura (STALLINGS, 2013).

Algoritmos para criptografia simétrica são implementados de forma que não seja praticável a dedução da mensagem ou decifragem do *ciphertext* a partir de apenas o próprio *ciphertext* e conhecimento sobre funcionamento do algoritmo de criptografia. Isso permite que não seja necessário manter o algoritmo secreto, apenas a chave, e é o que torna a criptografia simétrica viável para uso em larga escala (STALLINGS, 2013).

Um esquema criptográfico é dito incondicionalmente seguro se é impossível decriptar um *ciphertext* não importando quanto tempo um atacante possui disponível ou quanto *ciphertext* ele possui em mãos porque não existe informação suficiente para que seja

possível determinar unicamente o *plaintext* correspondente (STALLINGS, 2013). Existe apenas um algoritmo que satisfaz esse requisito, conhecido como One-time Pad, e ele não é prático para uso geral. Em vez disso, os algoritmos que são utilizados na prática buscam apenas ser computacionalmente seguros. Um esquema criptográfico é considerado computacionalmente seguro se o custo para quebra da criptografia excede o valor da informação criptografada ou se o tempo requerido excede o tempo útil da informação (STALLINGS, 2013). Isso significa que as cifras produzidas por algoritmos simétricos no geral não são indecifráveis, mas apenas que o custo computacional e tempo envolvido em decifrar uma dessas cifras é praticamente inviável.

As primeiras formas de algoritmos simétricos funcionavam através de operações de permutação e substituição sobre o *plaintext* (STALLINGS, 2013). Um dos primeiros algoritmos largamente utilizados e padronizados de criptografia simétrica foi o Data Encryption Standard (DES), que funcionava através de várias séries de permutação e substituição. Esse algoritmo foi amplamente utilizado até o início dos anos 2000 (STALLINGS, 2013).

2.1.1 AES

O Advanced Encryption Standard (AES) foi publicado em 2001 e veio a substituir o DES como o algoritmo de criptografia simétrica mais largamente utilizado (STALLINGS, 2013). O AES, assim como virtualmente todos os algoritmos criptográficos utilizados na prática, faz uso de operações aritméticas para o seu funcionamento (STALLINGS, 2013).

O AES é o protocolo criptográfico padrão da indústria que protege informação sensível de ataques de força bruta tradicional. Com a tecnologia atual, o AES é computacionalmente seguro contra ataques de força bruta e é usado em um incontável número de aplicações, desde para a proteção de informações em agências governamentais como para a proteção de dados de usuários comuns em aplicações em nuvem (HOUGEN, 2021). Ele é usado para proteger inúmeros dispositivos, applicações e redes em uso hoje em dia, incluindo discos rígidos, WiFi em redes locais, armazenamento em nuvem, navegadores de internet e comunicação segura com protocolos como Transport Layer Security (TLS), que utilizado para encriptar conexões e transações na navegação na internet (DANIEL, 2021).

2.2 CRIPTOGRAFIA DE CHAVE PÚBLICA

O desenvolvimento da criptografia de chave pública, ou criptografia assimétrica, é a maior e talvez a única verdadeira revolução na história da criptografia (STALLINGS, 2013). Ela foi uma radical mudança com relação ao que existia anteriormente, sendo baseada em funções matemáticas e fazendo uso de um par de chaves, em contraste aos anteriores métodos mais primitivos que faziam uso de permutações e substituições e à criptografia simétrica que utiliza uma única chave para encriptar e decriptar (STALLINGS, 2013). Algoritmos de criptografia de chave pública fazem uso de problemas matemáticos computacionalmente custosos para garantir a segurança do esquema e impedir que a chave privada possa ser calculada através de força bruta (STALLINGS, 2013).

Na criptografia de chave pública, uma das chaves do par é mantida de forma secreta pelo seu proprietário, chamada de chave privada, enquanto a outra é disponibilizada publicamente, chamada de chave pública. Dados que são cifrados pela chave privada apenas podem ser decifrados pela chave pública e vice-versa (STALLINGS, 2013). Da mesma forma que ocorre com a chave única utilizada na criptografia simétrica, é requisito indispensável que a chave privada esteja mantida de forma segura e seu acesso seja restrito aos indivíduos e entidades envolvidos na sua utilização para que operações com criptografia assimétrica sejam seguras (STALLINGS, 2013).

Utilizando uma chave pública, é possível enviar mensagens encriptadas para o seu proprietário. Uma vez que uma cifragem realizada com uma chave pública apenas pode ser decifrada pela sua chave privada correspondente, apenas o proprietário do par de chaves será capaz de ler uma mensagem cifrada com a sua chave pública. Isso é a criptografia assimétrica aplicada ao mesmo principal objetivo das técnicas criptográficas anteriores a ela, a comunicação segura.

Na prática, no entanto, comparando criptografia simétrica com assimétrica, a criptografia assimétrica requer mais computações, tornando a criptografia assimétrica não muito apropriada para grandes volumes de dados (IBM, 2022). No entanto, é possível, e comum, o uso de criptografia assimétrica para a negociação de chaves simétricas, que então podem ser utilizadas para encriptar dados adicionais (IBM, 2022).

2.2.1 Assinaturas Digitais

O mais importante desenvolvimento do trabalho com criptografia de chave pública, no entanto foi a assinatura digital. Assinaturas digitais provém um conjunto de capacidades de segurança que seriam difíceis de implementar de outra forma (STALLINGS, 2013).

Da mesma forma que você pode usar a chave pública de alguém para criptografia e ter aquela pessoa decriptar a sua mensagem com a chave privada dela, você também pode usar a sua chave privada para criptografia e ter que outros usem a sua chave pública para decodificação. Embora isso não seja desejável quando você estiver criptografando dados sensíveis, é uma parte importante da assinatura digital de qualquer dado (IBM, 2022). Um algoritmo de geração de assinatura digital recebe como entrada os dados a serem assinados e a chave privada do assinador e produz como saída uma assinatura digital (STALLINGS, 2013). Em vez de criptografar os dados em si, normalmente é criado um *hash* unidirecional dos dados e em seguida esse *hash* é criptografado com a sua chave privada. O resultado, junto de outras informações como o algoritmo de *hashing*, é conhecido como assinatura digital (IBM, 2022).

Algoritmos de verificação de assinatura recebem como entrada a chave pública, a assinatura em si e os dados originais que foram assinados e produzem como saída um indicador que afirma se a assinatura é válida ou não (STALLINGS, 2013). Para validar a validade da assinatura, o algoritmo primeiro usa a chave pública do signatário para decriptar a assinatura digital. Então é utilizado o mesmo algoritmo de *hash* que gerou o *hash* original para gerar um novo *hash* unidirecional a partir dos mesmos dados. Finalmente, o receptor compara os dois valores de *hash*. Se coincidirem, os dados não foram alterados desde que foram assinados. Se os *hashes* não coincidirem, os dados podem ter sido adulterados desde que foram assinados pela primeira vez ou a assinatura digital pode ter sido criada com uma chave privada que não corresponde à chave pública apresentada pelo signatário (IBM, 2022).

Assinaturas digitais validam a identidade do autor da assinatura, porque só podem ser geradas pela chave privada, garantem a integridade dos dados assinados, porque qualquer adulteração fará com que a verificação da assinatura falhe, e podem ser verificadas por qualquer entidade sem a necessidade de quebra do sigilo da chave privada, porque o processo de verificação não envolve a chave privada, somente a chave pública (STALLINGS, 2013).

Acordos e transações que antes eram assinados em papel e entregues fisicamente estão agora sendo substituídos por documentos e fluxos de trabalho totalmente digitais à medida que mais negócios são conduzidos on-line. Atores maliciosos que querem roubar ou manipular dados para benefício próprio frequentemente presentes sempre que dados preciosos ou sensíveis são compartilhados. Para minimizar o risco de adulteração de documentos por partes maliciosas, as empresas devem ser capazes de verificar e autenticar que esses documentos comerciais, dados e comunicações críticos são confiáveis e entregues com segurança. Isso um dos fatores que torna assinaturas digitais importantes. (PAUL, 2017)

Assinaturas digitais possuem aplicações primariamente tecnológicas, como o funcionamento de protocolos como o HTTPS, mas também podem possuir valor jurídico. No Brasil, a lei Nº 14.063, de 23 de Setembro de 2020 dispõe sobre o uso de assinaturas eletrônicas em interações com entes públicos, em atos de pessoas jurídicas e em questões de saúde e sobre as licenças de softwares desenvolvidos por entes públicos, com o objetivo de proteger as informações pessoais e sensíveis dos cidadãos.

Você pode comparar a significância de uma assinatura digital com o de sua assinatura manuscrita. Uma vez que você tenha assinado os dados, é difícil negar que o tenha feito mais tarde. Porém, isso pressupõe que a chave privada não foi comprometida de alguma forma. As assinaturas digitais tornam difícil para o assinante negar ter assinado os dados (IBM, 2022).

2.2.2 Certificados Digitais

Certificados digitais são a principal forma de distribuição de chaves públicas. Certificados digitais consistem de uma chave pública, dados de identificação do proprietário da chave, e todas essas informações digitalmente assinadas por uma entidade que é confiada (STALLINGS, 2013). Essa entidade, conhecida como autoridade certificadora (AC), é geralmente uma agência governamental ou instituição financeira que é confiada por usuários. Um usuário pode apresentar a sua chave pública para a autoridade de uma forma segura e obter um certificado. Esse certificado então é publicado e qualquer um precisando da chave pública do usuário pode obtê-la a partir do seu certificado. Se a autoridade certificadora que assinou o certificado é confiada por quem está em busca daquela chave pública, então o certificado e como consequência a chave pública presente nele também serão considerados confiáveis (STALLINGS, 2013).

ACs podem emitir certificados para outras ACs para delegar o trabalho de emissão para entidades finais. Dessa forma, uma hierarquia de certificados é criada para verificar a validade de um certificado. Essa hierarquia é conhecida como cadeia de confiança. Uma cadeia de confiança consiste de uma âncora de confiança, a AC raiz, que é a entidade que deve ser de confiança para os usuários finais, um número de camadas de ACs intermediárias que tiveram seus certificados emitidos pela AC raiz, ACs finais que tiveram seus certificados emitidos por uma AC intermediária e as entidades finais que recebem seus certificados a partir das ACs finais (SSL.COM, 2021).

Se existe uma grande comunidade de usuários, não é prático que todos façam solicitações para a mesma AC raiz. Porque a AC assina certificados, cada participante deve obter uma cópia da chave pública da AC para verificar assinaturas e essa obtenção deve ser realizada de forma segura para que o usuário possua confiança nos certificados associados. Portanto, com muitos usuários, é mais prático existir um conjunto de ACs, cada qual provê a sua chave pública para uma fração de usuários (STALLINGS, 2013).

Agora, vamos supor que existe um certo usuário A que confia apenas numa certa AC raiz X e um outro usuário B que obteve seu certificado de uma AC raiz Y. A princípio o certificado de B é inútil para A. No entanto, se as duas ACs, X e Y, tiverem seguramente criado um vínculo de confiança entre si, assinando as chaves uma da outra de forma que esse vínculo é verificável por A, então A também deverá confiar nos certificados que fazem parte da cadeia de Y, incluindo B (STALLINGS, 2013). Dessa forma, mesmo com múltiplas ACs raiz, não é necessário que todos os usuários confiem individualmente em todas elas.

2.2.2.1 X.509

X.509 é o padrão da International Telecommunication Union para certificados de chave pública. X.509 é um padrão amplamente utilizado porque a estrutura de certificados e os protocolos de autenticação definidos no X.509 são usados numa variedade de contextos como Secure/Multipurpose Internet Mail Extensions (S/MIME), utilizado para a assinatura de e-mails, Internet Protocol Security (IPsec), utilizado para o funcionamento de comunicação segura sobre o Internet Protocol, e Transport Layer Security, necessário para o funcionamento de HTTPS (STALLINGS, 2013).

X.509, além de definir um padrão para certificados, ele também define períodos de validade para certificados e a possibilidade de revogação de certificados (ITU, 2019). Isso faz com que usuários que tiveram suas chaves privadas comprometidas possam evitar serem personificados por um terceiro. ACs devem manter listas contendo todos os certificados revogados por aquela AC. Essas listas de certificados revogados (LCRs) são assinados pela própria AC e disponibilizadas publicamente em um repositório (ITU, 2019).

Quando um usuário recebe um certificado X.509, ele é capaz de verificar a assinatura da AC, a data de expiração do certificado e a LCR para garantir que aquele certificado é válido e aquela chave pública pode ser confiada.

2.2.3 Infraestrutura de Chaves Públicas

O RFC 4949 define infraestrutura de chaves públicas (*public-key infrastructure*, PKI) como o conjunto de hardware, software, pessoas, políticas, e procedimentos necessários para criar, administrar, armazenar, distribuir e revogar certificados digitais baseados em criptografia assimétrica.

Segundo o RFC, as funções centrais de uma PKI são registrar usuários e emitir seus certificados de chave pública, para revogar os certificados quando necessário, e arquivar os dados necessários para validar certificados em um momento muito posterior. Ele também recomenda que pares de chaves sejam gerados pelo próprio usuário e não por entidades do PKI como ACs para ajudar a manter a integridade do sistema criptográfico.

A Internet Engineering Task Force (IETF) também definiu a Public Key Infrastructure X.509 (PKIX), que é um modelo formal baseado no X.509 que busca ser adequado para a implantação de uma arquitetura baseada em certificados na Internet no RFC 2527.

Os componentes chaves do modelo PKIX são os seguintes:

- Entidade final: Um termo genérico usado para se referir a usuários finais, dispositivos ou qualquer outra entidade que pode ser identificada por um certificado de chave pública.
- Autoridade Certificadora (CA): A emissora de certificados e de LCRs. Pode também realizar uma série de funções administrativas, mas elas são geralmente delegadas para Autoridades de Registro.

- Autoridades de Registro: Um componente opcional que pode assumir um número de funções administrativas de uma AC, geralmente funções como registrar e aprovar informações de entidades finais.
- Emissor da LCR: Um componente opcional para o qual uma AC pode delegar a emissão de LCRs.
- Repositórios: Onde certificados e/ou LCRs são armazenadas para que entidades finais possam obter acesso.

2.3 HARDWARE SECURITY MODULE

Um Hardware Security Module (HSM) é um dispositivo dedicado para gerenciamento e manutenção de chaves criptográficas digitais. HSMs são módulos projetados especificamente para proteger o ciclo de vida de chaves através de poderosos mecanismos de segurança (AVI NETWORKS, 2022).

HSMs são fabricados seguindo uma série de padrões de segurança que os torna confiáveis para o armazenamento de chaves criptográficas (AVI NETWORKS, 2022). Eles incluem até múltiplos chips de processadores criptográficos para veloz execução de operações criptográficas, um sistema operacional focado em segurança e acesso limitado à rede através de uma interface de rede estritamente controlada. Partes especiais do hardware são designadas especificamente para gerar entropia para produzir chaves criptográficas aleatórias de boa qualidade rapidamente (AVI NETWORKS, 2022).

Alguns mecanismos que tornam HSMs confiáveis são funcionalidades como a geração de registros ou logs de operações e acesso, emissão de alertas sobre possíveis tentativas de interferência ou adulteração ou ainda a capacidade de destruir objetos criptográficos ou se tornar inoperável em casos de interferência externa (AVI NETWORKS, 2022). Adicionalmente, tentar interferir ou acessar indevidamente um HSM por si só já é uma tarefa extremamente difícil devido aos vários níveis de proteção presentes tanto no software que é executado pelo HSM quanto pelo seu próprio hardware (AVI NETWORKS, 2022).

HSMs são essenciais para o funcionamento de infraestruturas de chave pública e vários outros tipos de infraestruturas essenciais que dependem de altos níveis de segurança (AVI NETWORKS, 2022). Para estes tipos de aplicação, HSMs podem até ser transformados em clusters para maior disponibilidade (AVI NETWORKS, 2022).

3 METODOLOGIA E DESENVOLVIMENTO

Esta seção transcorrerá sobre as tecnologias, métodos e sobre as decisões e soluções tomadas durante o desenvolvimento do projeto.

3.1 TECNOLOGIAS

O sistema web foi desenvolvido na linguagem Java utilizando o framework Spring, banco de dados MySQL e arquitetura REST. A documentação do sistema web foi gerada no formato OpenAPI 3.0. O projeto do sistema foi criado utilizando o Apache Maven para gerenciamento de dependências. O sistema foi executado e testado exclusivamente no sistema operacional Ubuntu 20.04.4 LTS. Foi utilizado um Java Development Kit versão 11, como especificado no arquivo "*pom.xml*" do projeto.

Nas subseções a seguir, serão apresentadas e justificadas algumas das tecnologias utilizadas para o desenvolvimento do projeto.

3.1.1 A Linguagem Java e o Framework Spring

A linguagem Java foi escolhida para o desenvolvimento do sistema por dois motivos. Primeiro, a familiaridade e experiência do autor deste trabalho com a sua utilização. Segundo, a enorme quantidade de suporte disponível para desenvolvimento web na linguagem, tanto na biblioteca padrão quanto em diversos frameworks e bibliotecas externas.

O framework selecionado para o desenvolvimento do sistema foi o framework Spring. O Spring é um framework desenvolvido para prover um modelo compreensivo de programação e configuração para aplicações Java modernas, com extenso suporte para aplicações web. Atualmente, o Spring é o framework Java mais utilizado no mundo de acordo com o Snyk.

O Spring provê funcionalidades como injeção de dependências, mocking e gerenciamento de contextos para testes, fáceis mecanismos de acesso a banco de dados e um framework web completo que permite a separação entre aspectos de lógica de aplicação, tratamento de respostas HTTP, filtros sobre requisições e um sistema de autenticação e autorização. Todas essas funcionalidades são extremamente úteis ou necessárias para o desenvolvimento da aplicação objetivo, o que faz do Spring uma escolha ideal.

3.1.2 OpenAPI

A OpenAPI Specification é uma especificação para interfaces de serviços web RESTful originalmente baseada na Swagger Specification. Ela é uma especificação aberta dentro da OpenAPI Initiative dirigida pela comunidade que define um padrão agnóstico a linguagens de programação de fácil interpretação por tanto humanos e computadores.

Por ser uma forma de especificação livre e amplamente utilizada, ela foi escolhida como o formato para produção da documentação da API do sistema desenvolvido.

3.1.2.1 springdoc-openapi

O springdoc-openapi é uma biblioteca que ajuda na automatização da geração da documentação da API de sistemas web Spring. Essa biblioteca examina uma aplicação em execução para inferir semânticas da API baseando-se em configurações do Spring e nas definições de classes e métodos.

Por si só, o springdoc-openapi realiza uma grande porção do trabalho, mas ele não é capaz de obter informações sobre certas partes do sistema, como, por exemplo, o esquema de autenticação e autorização e, no caso da implementação atual do sistema, o formato do corpo de resposta enviado pelo servidor, que é construído dinamicamente. No entanto, isso não é um problema, porque a biblioteca também permite o uso de configurações Spring e de anotações sobre a métodos e classes para complementar a interpretação da API.

Com essa ferramenta, foi possível produzir uma documentação completa da API do sistema sem grandes dificuldades. No caso de que uma possível versão do sistema leve a alterações na API, a atualização da documentação fazendo uso do springdoc-openapi é algo trivial.

3.2 ANÁLISE DO PROBLEMA

Inicialmente foram estabelecidos os requisitos do sistema. As principais operações identificadas são a criação de chaves e realização de assinaturas com essas chaves. Para isso,

seria necessário um sistema que restrinja o acesso às chaves aos seus criadores. Além disso, as chaves precisam ser mantidas de forma segura.

Para implementar essas funcionalidades básicas e seus requisitos, durante o desenvolvimento do sistema, algumas decisões precisaram ser tomadas com relação a detalhes importantes do seu funcionamento, principalmente detalhes que dizem respeito a segurança. As subseções a seguir enumeram e discorrem sobre as questões consideradas mais importantes.

3.2.1 Usuários do sistema

Foi identificada a necessidade de três tipos de usuários para o sistema, dois tipos usuários internos do sistema e um usuário externo ao sistema: usuários comuns, o administrador do sistema e o administrador do banco de dados.

Usuários comuns são aqueles que podem utilizar o sistema para gerenciar chaves e realizar assinaturas. Eles possuem acesso aos *endpoints* relacionados a chaves, que os administradores não possuem.

A forma de autenticação escolhida para os usuários foi a combinação de nome de usuário e uma senha conhecida apenas pelo próprio usuário devido à sua efetividade e facilidade de aplicação. Autenticação utilizando biometria foi considerada, mas descartada devido à possível complexidade envolvida para a implementação neste trabalho. Autenticação utilizando assinatura digital também foi considerada, mas não foi ultimamente implementada por causa do esquema de armazenamento seguro desenvolvido descrito na seção 3.2.2, que é uma solução que depende de uma senha para o usuário.

3.2.1.1 Configuração do banco de dados e usuários administradores

Um requisito de sistema julgado importante foi a separação do papel de administrador do banco de dados e de administrador do sistema. O administrador do banco de dados gerencia exclusivamente o banco de dados do sistema enquanto a configuração de outras partes do sistema, como, em especial, as credenciais para comunicação com o HSM, são responsabilidades do administrador do sistema. Isso tem como objetivo impossibilitar que o administrador do sistema seja capaz de associar pares de chaves a seus usuários, uma vez que ele não terá acesso ao banco de dados que faz essa associação, dessa forma não sendo

capaz de realizar um ataque a um usuário específico. Ao mesmo tempo isso impede que o administrador do banco de dados tenha qualquer tipo de acesso às chaves, uma vez que ele não conhece as credenciais do HSM.

A intenção inicial era que a configuração do banco de dados fosse realizada pelo administrador do banco de dados em um *endpoint* do sistema que armazenaria as credenciais do banco criptografadas com uma chave gerada a partir de uma senha informada pelo administrador. Porém, uma implementação desse tipo, com configuração do banco de dados em tempo de execução, se mostrou difícil devido a limitações do Spring. Esse problema foi encontrado tardiamente durante o desenvolvimento, então, em vez de uma mudança brusca como uma completa troca de framework, uma nova solução foi buscada.

A solução encontrada foi armazenar a configuração do banco de dados em um arquivo de texto que é lido pelo sistema durante a sua inicialização. O arquivo deve possuir o nome "*settings.json*" e ser inserido no diretório */etc/psc/*. Esse arquivo deve apenas possuir permissões de leitura e escrita para o usuário responsável pelo sistema. Por fim, a pessoa que possui acesso ao usuário do sistema operacional que pode configurar o arquivo e executar o sistema deve ser o administrador do banco de dados. O sistema analisará as permissões do arquivo e rejeitará arquivos com permissões de leitura ou escrita diferentes.

Como consequência dessa decisão, apesar de o papel de administrador do banco de dados ser real e necessária para o funcionamento do sistema, esse tipo de usuário não é tratado internamente pelo sistema, uma vez que ele nunca se comunica com o sistema através de qualquer *endpoint*, ele é um usuário totalmente externo ao sistema.

3.2.2 Armazenamento seguro no banco de dados

Algumas informações precisam ser armazenadas de forma segura no banco de dados. Não apenas a senha de autenticação dos usuários, mas também os identificadores das chaves no HSM e as próprias credenciais do HSM.

Para as senhas, como em outros sistemas que fazem autenticação utilizando senhas, apenas um *hash* da senha é armazenado. A autenticação, mesmo com apenas um *hash* da senha é possível porque o sistema pode gerar o *hash* da senha informada pelo usuário e compará-la com o *hash* que possui armazenado. O algoritmo de *hash* escolhido foi o bcrypt, que é o algoritmo padrão utilizado no OpenBSD, que possui uma implementação em Java

facilmente integrável com o sistema de autenticação do Spring. O bcrypt requer um valor para ser utilizado como *salt*, protegendo o sistema de formas comuns de ataque sobre algoritmos de *hashing*.

Quanto às outras informações, o acesso deve ser exclusivo do usuário às quais o dado é associado. Para que isso fosse possível, a forma encontrada foi encriptar os dados com alguma informação de exclusivo conhecimento do usuário, isto é, a própria senha do usuário.

Porém, buscando evitar possíveis ataques critanalíticos que possam resultar da escolha de uma senha ruim pelo usuário, uma transformação foi escolhida para a senha. Em vez de a senha ser utilizada como chave diretamente, a chave criptográfica para é a concatenação do nome do usuário com a sua senha, e o resultado dessa concatenação novamente concatenado com a sequência "PSC". Dessa forma, a chave é um valor que só é derivável a partir de uma informação que apenas o usuário conhece, a sua senha, e a sua segurança não depende da qualidade da senha escolhida pelo usuário.

O algoritmo simétrico de criptografia escolhido foi o AES, que é o protocolo de criptografia padrão da indústria e inquebrável através de ataques de força bruta com a tecnologia atual (HOUGEN, 2021). Esse algoritmo não aceita chaves de qualquer tamanho, portanto o tamanho do resultado da concatenação é ajustado para se adequar ao algoritmo. Bits são removidos ou zeros são adicionados até que o tamanho da chave se torne 256 bits.

3.2.3 Armazenamento de Chaves

As chaves em si são armazenadas em um HSM cujas credenciais são configuradas pelo usuário administrador do sistema. Porém, o sistema precisa armazenar tanto as credenciais de comunicação com o HSM quanto identificadores das chaves no HSM para poder acessá-las.

As credenciais do HSM que são configuradas pelo administrador são armazenadas no banco de dados criptografadas utilizando a chave descrita na seção 3.2.2. O banco de dados armazena as seguintes informações sobre os pares de chaves:

- um ID sequencial que é utilizado como chave primária no banco de dados;
- o identificador único da chave pública no HSM;
- o identificador único da chave privada no HSM;
- o algoritmo do par de chaves;
- um identificador único para o par de chaves, gerado pelo próprio sistema;

- o nome da chave, atribuído pelo seu usuário proprietário durante a criação;
- o usuário que é proprietário do par de chaves.

Dentre esses dados, o identificador único da chave privada do par de chaves é armazenado criptografado com o mesmo método descrito na seção 3.2.2.

Esse esquema, juntamente da separação de papéis entre administrador do sistema e administrador do banco de dados, como explicado na seção 3.2.1.1, faz com que não seja possível que o gerenciador do banco de dados obtenha acesso às credenciais do HSM a partir das credenciais do banco de dados. O esquema de criptografia utilizado sobre o identificador da chave privada é uma camada adicional de segurança para impedir que o administrador do sistema, que possui conhecimento das credenciais do HSM, possa diretamente associar um usuário com suas chaves caso venha a obter acesso às informações do banco de dados de alguma forma ou que o administrador do banco de dados possa fazer o mesmo caso venha a obter acesso ao HSM de alguma forma.

No entanto, essa solução não é perfeita porque assume que os dois administradores não podem trabalhar juntos para comprometer as chaves dos usuários. É razoável assumir que o administrador do sistema deve ser confiável, porque é ele quem possui acesso direto às chaves privadas dos usuários no HSM. Com essa assunção, a solução aparenta ser consideravelmente segura, mas um sistema de auditoria bem implementado poderia remover a necessidade de tal assunção.

3.2.4 Autenticação e autorização

Evidentemente, seguindo a descrição da seção 3.2.2 sobre a forma escolhida para proteger informações sensíveis, o sistema precisa fazer uso de uma combinação de nome de usuário e senha para autenticação. Isso deve ser aplicado para todas as operações no sistema que envolvem utilizar ou criar uma chave. Porém, isso é muito inconveniente, principalmente para múltiplas requisições em sequência, como, por exemplo, criar um par de chaves e logo em seguida assinar dados. Uma alternativa para a autenticação que tem como objetivo evitar o constante reenvio de credenciais é o uso de um *token* de acesso. O usuário se autenticaria num *endpoint* de autenticação e obteria o *token* que possuiria dentro de si informações como o tipo do usuário (usuário comum ou administrador) para ser usado pelos filtros de autorização do sistema.

No entanto, à primeira vista, isso não é possível. Voltando à descrição da seção 3.2.2, o sistema utiliza as credenciais do usuário, incluindo sua senha, para realizar operações no banco de dados. Uma vez que todas as operações dependem das credenciais do usuário para encriptar ou decriptar alguma informação sobre chaves, o usuário precisaria se autenticar em qualquer chamada feita para o sistema, não só inviabilizando a possibilidade de um esquema de autenticação com um *token* de acesso, mas também tornando o uso do sistema bastante inconveniente.

A solução encontrada para esse problema é inserir essas credenciais encriptadas no próprio *token* utilizando uma chave que é de conhecimento exclusivo do sistema. Quando o sistema é inicializado, essa chave é gerada e ela é utilizada enquanto o sistema está em execução para assinar os *tokens* e para encriptar e decriptar as credenciais do usuário que são inseridas nos *tokens*. O formato da chave escolhido foi uma chave AES de 256 bits. A chave do sistema é gerada novamente quando o sistema é reinicializado, mas ela também pode ser atualizada com uma solicitação do administrador do sistema.

Dessa forma, o sistema é capaz fazer uso de um esquema de autenticação baseado *tokens*, para evitar a inconveniência de exigir as credenciais do usuário em todas as operações, e ao mesmo tempo manter o esquema de criptografia que protege as informações do banco de dados.

O sistema, então, quando recebe uma solicitação de autenticação, concatena o nome do usuário, sua senha e a *string* “PSC”, encripta os bytes do resultado dessa concatenação utilizando a sua chave interna via AES e converte o resultado para Base64. O resultado final é conhecido internamente no sistema como “*access key*” ou “chave de acesso”. Apesar de apenas um *hash* da senha do usuário ser mantido no banco de dados, a geração da chave de acesso é possível porque é necessário que o usuário informe sua senha durante a autenticação.

Para fazer uso da chave de acesso, o sistema decodifica o Base64, decripta essa decodificação utilizando a chave do sistema e em seguida realiza a operação desejada. Isso ocorre quando é necessário registrar informações criptografadas no banco de dados, ou obter informações criptografadas armazenadas no banco de dados, como, por exemplo, durante o processo de criação de chaves, em que a chave de acesso é utilizada para encriptar o identificador da chave primária. Esses processos de criptografia envolvendo a chave de acesso utilizam o AES.

3.2.4.1 O Token de Acesso

O *token* de acesso utilizado emitido para o usuário quando ele se autentica no sistema é um JSON Web Token (JWT). JWTs foram definidos no RFC 7519 e são formas compactas para transportar alegações ("*claims*") entre duas partes comunicantes. JWTs podem ser assinados digitalmente para garantir sua integridade. JWTs são o formato ideal de *token* para o sistema porque podem carregar a chave de acesso em suas *claims* e podem ser assinados digitalmente pelo sistema para garantir a integridade dessa chave de acesso. Adicionalmente, JWTs também suportam uma data de expiração, o que é útil porque não é desejável que um *token* emitido uma única vez seja permanentemente aceitável pelo sistema, é preciso que o usuário se autentique de novo periodicamente para assegurar que ele é realmente aquele quem está acessando o sistema.

O JWT emitido pelo sistema para o usuário durante o processo de autenticação, possui duas *claims*: "*accessKey*", que armazena a chave de acesso do usuário, e "*roles*", que armazena o tipo do usuário (administrador ou usuário comum). O JWT possui um tempo de expiração e será rejeitado pelo sistema 8 horas após a sua emissão, necessitando que o usuário se autentique novamente.

O JWT é assinado com a chave do sistema utilizando HMAC-SHA256. No caso de atualização da chave do sistema, seja por reinicialização do sistema ou por requisição do administrador, JWTs assinados com a chave anterior serão rejeitados pelo sistema mesmo que não tenham atingido seu prazo de expiração, uma vez que a chave que foi utilizada para assiná-los foi perdida, o que não é grande problema porque o usuário pode simplesmente se autenticar novamente e obter um novo JWT.

De posse do JWT, o usuário pode fazer requisições sobre recursos que possuem mecanismo de autorização, como os recursos relacionados a pares de chaves no caso de usuários comuns ou recursos de configuração do sistema no caso do usuário administrador.

3.2.4.2 HTTPS

Sistemas de autenticação nesse estilo não são necessariamente seguros. Para garantir a segurança é necessário fazer uso de HTTPS para criptografar o corpo dos pacotes e impedir que o *token* de autenticação seja transportado em *plaintext*.

Felizmente, o Spring permite a configuração de HTTPS no sistema através do seu arquivo de configuração "*application.properties*". Como será explicado na seção 4.3, o sistema é executado fazendo uso do próprio Spring Boot, o que permite que os valores do arquivo de configuração sejam alterados por variáveis de ambiente. Dessa forma, o responsável pela execução do sistema, que, como explicado na seção 3.2.1.1, será o administrador do banco de dados, pode realizar essa configuração.

As variáveis de ambiente disponíveis são as seguintes:

- SERVER_SSL_ENABLED, que deve possuir valor "*true*" para tornar HTTPS disponível e possui valor padrão "*false*";
- SERVER_SSL_KEY_STORE_TYPE, para identificar o formato do arquivo que armazena a chave e os certificados confiáveis, por exemplo "JKS" ou "PKCS12";
- SERVER_SSL_KEY_STORE, para o caminho para o arquivo;
- SERVER_SSL_KEY_STORE_PASSWORD, para a senha do arquivo;
- SERVER_SSL_KEY_PASSWORD, para a senha da chave;
- SERVER_SSL_KEY_ALIAS, para o nome da chave.

3.3 DESENVOLVIMENTO DO SISTEMA

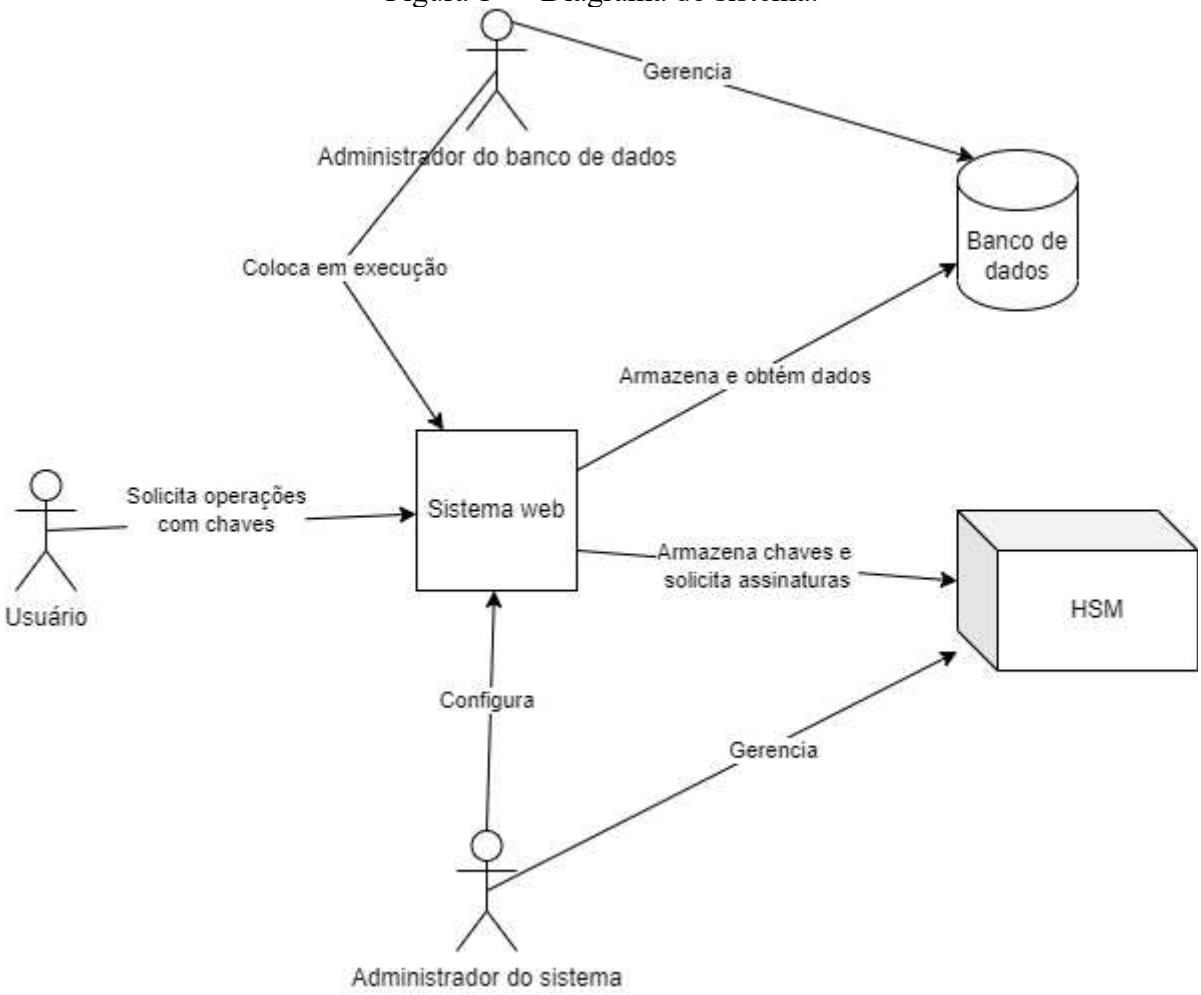
Com os requisitos do sistema delineados, um esqueleto da API do sistema web foi construído. A partir desse esqueleto, foi implementada um protótipo parcial do sistema. Esse protótipo foi testado através de testes de unidade e testes de integração automatizados implementados na linguagem de programação escolhida e através de testes manuais sobre uma implantação local do sistema utilizando ferramentas como o Postman para enviar requisições HTTP.

A partir do protótipo inicial, o desenvolvimento se seguiu através de ciclos em que a API do sistema era atualizada, o protótipo era refinado, recebendo funcionalidades adicionais, e os testes eram ampliados. Esse processo cíclico ocorreu paralelamente à análise descrita na seção 3.2, de forma que o sistema evoluiu lentamente de um protótipo inicial simples com chaves geradas em software e sem qualquer tipo de esquema de autenticação até a sua forma atual.

4 RESULTADOS

Nesta seção, o sistema que foi desenvolvido no trabalho será apresentado em detalhes. A figura 1 mostra as relações entre as partes do sistema de uma forma simplificada.

Figura 1 — Diagrama do sistema.



Fonte: Elaboração própria

4.1 API E FUNCIONALIDADES DO SISTEMA

No sistema em execução, a API do sistema pode ser obtida em /v3/api-docs. A documentação também está disponível no repositório do GitHub em https://github.com/guilhermesartori/TCC_PSC/blob/master/api-docs.json. A documentação

encontrada neste arquivo é padrão OpenAPI 3.0, na forma de um JSON, gerada pela biblioteca springdoc-openapi, como descrito na seção 3.1.2.

Um outro *endpoint* que apresenta a documentação do sistema é /swagger-ui.html. Esse *endpoint* apresenta uma interpretação da documentação OpenAPI em uma página web que também pode ser utilizada para enviar chamadas de teste para os recursos do sistema. O formato da documentação nessa página web é fácil de entender para aqueles que não conhecem o formato OpenAPI.

Adicionalmente, como o formato OpenAPI é amplamente suportado, várias ferramentas existem que podem fazer a interpretação da documentação para mais fácil entendimento, como o Postman, o ApiBldr, o Apicurio, plugins em IDEs como o IntelliJ IDEA, entre outros.

A tabela 1 abaixo apresenta os *endpoints* do sistema web, os métodos HTTP suportados por cada um e uma sucinta descrição (com exceção dos *endpoints* citados acima para acessos à documentação da API).

Tabela 1 – Descrição dos recursos da API do sistema

Endpoint	**Métodos HTTP**	**Descrição**
/user	POST, GET	Operações relacionadas a usuários
/user/{username}	GET	Operações relacionadas a um usuário específico. {username} é um placeholder para o nome do usuário especificado.
/login	POST	Operações de autenticação
/key	POST, GET	Operações relacionadas a pares de chaves
/key/{keyUniqueIdentifier}	GET, DELETE	Operações relacionadas a um par de chaves específico. {keyUniqueIdentifier} é um placeholder para o identificador único do par de chaves especificado.
/key/{keyUniqueIdentifier}/sign	POST	Operações relacionadas a geração de assinaturas
/key/{keyUniqueIdentifier}/verify-signature	POST	Operações relacionadas à verificação de assinaturas.
/system/admin-user	POST	Operação relacionada ao usuário administrador do sistema.
/system/hsm-config	PUT	Operação relacionada às

		credenciais do HSM utilizado pelo sistema.
/system/hsm-config/load	POST	Operação para carregamento de credenciais de HSM já inseridas no sistema.
/system/refresh-keys	POST	Operação para atualizar as chaves internas do sistema.

Fonte: Elaboração própria.

4.1.1 Autorização

A autorização é realizada no sistema através de um filtro de requisições do framework Spring. Requisições enviadas para recursos que requerem autorização devem incluir um cabeçalho HTTP Authorization contendo a sequência de caracteres “Bearer” seguida de um espaço e o JWT emitido para o usuário no recurso de login. O JWT armazena as autoridades do usuário, isto é, se ele é um usuário comum ou o administrador, e, através da decodificação do JWT e análise dessas autoridades, requisições são filtradas ou permitidas.

A tabela 2 demonstra os requisitos de autorização para os recursos e métodos HTTP que foram configurados no filtro de requisições do Spring.

Tabela 2 – Autorização nos recursos do sistema

Recurso	**Método HTTP**	**Autorização**
/user	GET	ADMINISTRATOR
/user/{username}	GET	ADMINISTRATOR
/key	POST	USER
/key/{uniqueIdentifier}	DELETE	USER
/key/{uniqueIdentifier}	GET	USER
/key/sign	POST	USER
/system/knet-config	PUT	ADMINISTRATOR
/system/refresh-key	POST	ADMINISTRATOR

Fonte: Elaboração própria

4.1.2 Detalhamento das operações

Esta seção apresentará em detalhes todas as operações sobre todos os *endpoints* do sistema, o formato dos corpos das requisições, os parâmetros de URL e parâmetros de *query*, os status HTTP de resposta, o corpo das respostas e detalhes sobre o funcionamento de cada operação.

4.1.2.1 POST /user

Este é o *endpoint* para registrar novos usuários comuns. O *endpoint* para registrar o usuário administrador está descrito na seção 4.1.2.11. Este *endpoint* recebe em seu corpo um JSON com os campos descritos na tabela 4. Porque esse *endpoint* é utilizado para registrar um novo usuário comum, ele não exige o cabeçalho Authorization. Os status de resposta e corpos de resposta estão na tabela 5.

Este *endpoint*, assim como vários outros que serão descritos em seguida, irão sempre retornar uma resposta com HTTP status 400 Bad Request se o sistema se encontra desconfigurado, como descrito na seção 4.2.

4.1.2.2 GET /user

Este é o *endpoint* para obtenção de informações dos usuários do sistema. A tabela 6 descreve o formato do *request* que deve ser enviado ao *endpoint* e a tabela 7 descreve as possíveis respostas do sistema. Este *endpoint* só pode ser acessado por um administrador autenticado.

4.1.2.3 GET /user/{username}

Este *endpoint* obtém as informações de um usuário específico. "*{username}*" representa o nome de algum usuário. As tabelas 8 e 9 descrevem respectivamente o formato do *request* enviado e as possíveis respostas. Assim como o *endpoint* em descrito na seção anterior, este *endpoint* só está disponível para usuários administradores.

4.1.2.4 POST /login

Este é o *endpoint* de autenticação do sistema. Aqui, um usuário informa suas credenciais e recebe o JWT utilizado para autenticação. O sistema gera o *hash* da senha informada e o compara com o *hash* armazenado no banco de dados para realizar a autenticação. As particularidades do *request* e da *response* estão nas tabelas 10 e 11.

4.1.2.5 POST /key

Usuários autenticados podem criar chaves através deste *endpoint*, fornecendo os parâmetros da chave em campos de um JSON. As tabelas 12 e 13 apresentam os detalhes do corpo da requisição e das respostas.

Após receber os campos, o sistema primeiramente se comunica com o HSM para realizar a criação do par de chaves. Se a criação ocorre com sucesso, o sistema cifra o identificador da chave privada obtido como descrito anteriormente na seção 3.2.3 e então armazena as informações do par de chaves no banco de dados.

O campo "*keyAlgorithm*" necessário no *request* aceita exclusivamente os valores "EC", "RSA" e "EDDSA", correspondendo respectivamente aos algoritmos Elliptic Curve Digital Signature Algorithm (ECDSA), Rivest–Shamir–Adleman (RSA) e Edwards-curve Digital Signature Algorithm (EdDSA), que são os únicos algoritmos de chave pública suportados pelo sistema. O campo "*keyParameter*" deve corresponder ao tamanho da chave, para RSA, ao nome da curva elíptica, para EC, ou a "ED448" ou "ED25519", para EDDSA.

4.1.2.6 GET /key

Este *endpoint* obtém os pares de chaves de um usuário autenticado. Ele também pode ser utilizado para obter um único um par de chaves a partir do nome que foi atribuído a ela na sua criação, enviando seu nome como parâmetro de *query*. As tabelas 14 e 15 detalham o *request* e as *responses*. Na *response*, a chave privada não é incluída.

4.1.2.7 DELETE /key/{keyUniqueIdentifier}

Este *endpoint* é utilizado para remover um par de chaves no sistema. Ele deve ser chamado pelo detentor do par de chaves especificado. As tabelas 16 e 17 detalham o *request* e as *responses*.

Deletar o par de chaves realizará não apenas a remoção das informações do par de chaves armazenadas no banco de dados, mas também removerá tanto a chave pública como a chave privada do HSM.

4.1.2.8 GET /key/{keyUniqueIdentifier}

Endpoint semelhante ao descrito na seção 4.1.2.6 quando utilizado informando um nome de chave. Este *endpoint*, no entanto, recebe o identificador único da chave no caminho da solicitação em vez de receber um parâmetro de *query*. A chamada deve ser feita pelo usuário proprietário do par de chaves. As tabelas 18 e 19 detalham o *request* e as *responses*.

4.1.2.9 POST /key/{keyUniqueIdentifier}/sign

Este é o *endpoint* para geração de assinaturas. Ele deve ser chamado pelo usuário detentor do par de chaves ao qual o identificador único esta relacionado. As tabelas 20 e 21 detalham a operação.

No corpo da resposta, o sistema retorna um JSON com a assinatura gerada, o identificador do par de chaves utilizado e a chave pública do par de chaves codificada em Base64. O envio da chave pública tem como intenção simplificar um pouco a verificação da assinatura se for desejado realizá-la sem utilizar o próprio sistema.

Para as assinaturas que são retornadas no JSON, foi adotado o padrão de assinatura definido no PKCS1 (RFC 3447) devido à sua simplicidade e aplicabilidade. Este padrão é o suficiente para uma demonstração e prova de conceito, além de ser a base para outros padrões como o PKCS7.

4.1.2.10 POST /key/{keyUniqueIdentifier}/verify-signature

Este *endpoint* realiza a verificação de uma assinatura. No *endpoint* de assinatura, o sistema envia junto da assinatura no corpo da resposta a chave pública e o identificador do par de chaves, de forma a permitir que a verificação de assinatura possa ser feita tanto usando este

endpoint junto do identificador único do par de chaves como externamente utilizando a chave pública informada. Este *endpoint* não requer autenticação. Detalhes nas tabelas 22 e 23.

O nome do algoritmo de assinatura em "*signatureAlgorithm*" deve seguir o padrão esperado pela implementação do Java Security utilizada, no caso, o Bouncy Castle. Para assinaturas realizadas sem *hashing*, "RSA", "ECDSA", "Ed448" ou "Ed25519" são os valores suportados. Para assinaturas com chaves EC ou RSA em que foi utilizado um algoritmo de *hash* sobre os dados, pode ser utilizado o nome do algoritmo de *hash* concatenado com "WithRSA" ou "WithECDSA", como, por exemplo "SHA256WithRSA". A verificação também pode ser feita aplicando o *hash* sobre os dados originais e enviando esse *hash* no campo "*base64EncodedData*" e indicando o nome do algoritmo sem o algoritmo de *hash* no campo "*signatureAlgorithm*", e isso se aplica também para casos de assinaturas EdDSA.

4.1.2.11 POST /system/admin-user

Este *endpoint* faz a criação do administrador do sistema. Ele é extremamente similar ao *endpoint* que cria usuários comuns. Seus detalhes são apresentados nas tabelas 24 e 25.

O administrador do sistema é único: não é possível criar um segundo administrador. Além disso, uma vez criado, o administrador não pode ter suas credenciais modificadas. O administrador do sistema não possui permissões relacionadas a criação de chaves, apenas relacionadas a recursos de configuração.

4.1.2.12 PUT /system/hsm-config

Com um administrador criado e autenticado no sistema, é possível configurar as credenciais do HSM que armazenará as chaves do sistema e este é o *endpoint* com essa responsabilidade. As tabelas 26 e 27 detalham seu funcionamento.

Na requisição, o corpo do campo "*parameters*" deve conter pares de valor e chave para os parâmetros, como, por exemplo, um campo "*ADDRESS_CONN*" com o endereço IP do HSM como valor.

Este *endpoint* pode ser chamado novamente pelo administrador para alterar as credenciais do HSM. Porém, se ocorrer uma troca de HSM após a criação de chaves, as

chaves que existiam no sistema se tornarão inutilizáveis a não ser que as chaves antigas sejam restauradas no novo HSM com os mesmos identificadores.

4.1.2.13 POST /system/hsm-config/load

Este *endpoint* é utilizado para fazer o carregamento de credenciais do HSM que já existem dentro do banco de dados. Mais informações são apresentadas na seção 4.2 e em suas subseções. Detalhes sobre o *endpoint* estão nas tabelas 28 e 29.

4.1.2.14 POST /system/refresh-keys

Este *endpoint* faz a atualização das chaves do sistema que são utilizadas para emitir os *tokens* de autenticação. Ele pode ser chamado pelo administrador a qualquer momento, porém chamá-lo faz com que usuários precisem se autenticar novamente mesmo que os *tokens* de acesso em sua posse estejam dentro do seu prazo de validade. Detalhes são apresentados nas tabelas 30 e 31.

4.2 CONFIGURAÇÃO

Para pleno funcionamento do sistema, é primeiro necessário que ele seja configurado. Essa configuração se trata da configuração do banco de dados, da configuração do usuário administrador e da configuração das credenciais do HSM que será utilizado para armazenar os pares de chave do sistema.

4.2.1 Configuração do banco de dados

A configuração do banco de dados se trata do arquivo "*settings.json*" descrito anteriormente na seção 3.2.1.1.

Este arquivo deve ser um arquivo de texto na forma de um JSON. Esse JSON deve possuir um único campo, "*databaseConfiguration*", que deve possuir como valor um JSON com os campos descritos na tabela 31. O arquivo possui esse formato para permitir que possíveis futuras versões do sistema tenham facilidade em extendê-lo.

Tabela 3 – Campos de "*databaseConfiguration*" no arquivo "*settings.json*"

Campo	**Descrição**
url	Endereço do esquema de banco de dados, incluindo protocolo e porta. Por exemplo: "jdbc:mysql://localhost:3306/openpsc".
username	Nome do usuário para conexão com o banco de dados
password	Senha do usuário para conexão com o banco de dados

Fonte: Elaboração própria

4.2.2 Administrador do Sistema

A presença do administrador do sistema é necessária para a execução do sistema normalmente. Requisições ao sistema são rejeitadas se o sistema não possui um administrador criado. Dessa forma, é necessário que seja utilizado o *endpoint* na seção 4.1.2.11. Isso pode ser realizado antes da abertura das portas do servidor para recebimento de requisições externas.

4.2.3 Credenciais do HSM

Essa configuração não é única e pode ser alterada pelo administrador a qualquer momento. As credenciais do HSM são armazenadas no banco de dados criptografadas com a chave de acesso do administrador, como descrito na seção 3.2.2. Da mesma forma que o usuário administrador, a configuração do HSM é necessária para a execução do sistema e o sistema rejeitará requisições feitas sem essa configuração e ela pode ser realizada através do *endpoint* descrito na seção 4.1.2.12.

A configuração do arquivo com as credenciais do banco de dados é prerrequisito para execução do sistema e a inicialização do sistema falhará caso o arquivo esteja ausente ou incorreto.

4.2.4 Inicialização de um sistema já configurado

Um sistema que já teve sua configuração realizada e está sendo inicializado novamente precisa de tratamento adicional. O sistema não consegue fazer o carregamento das

credenciais do HSM a partir do banco de dados sozinho porque elas são armazenadas encriptadas com a chave de acesso do administrador do sistema.

Por causa disso, é necessário fazer o uso do *endpoint* descrito na seção 4.1.2.13 para carregar as credenciais do HSM para a memória do sistema. Como o *endpoint* requer a autenticação do administrador, ele receberá o JWT que possui dentro de si a chave de acesso do administrador para decriptar as credenciais armazenadas no banco de dados.

4.3 COMPILAÇÃO E EXECUÇÃO

O sistema utiliza o Maven para gerenciamento de dependências, então é preciso utilizar comandos do Maven para gerar o arquivo Jar do sistema. Em uma máquina com o Maven instalado e utilizando o JDK 11, um comando "*mvn install -Dmaven.test.skip*" a partir do diretório raiz do projeto é suficiente para produzir o Jar. Ele será criado no diretório "*target*" que será criado a partir do diretório raiz do projeto.

O sistema é executado com o próprio Spring Boot. Para executá-lo, pode-se primeiramente exportar quaisquer variáveis de ambiente desejadas para alterar as configurações do servidor, como, por exemplo, as configurações de HTTPS descritas na seção 4.4.2 ou a porta do servidor. Essas informações de configuração estão disponíveis na Spring Framework Documentation. Com as configurações desejadas estabelecidas, um simples "*java -jar*" com o caminho para o Jar será suficiente para colocá-lo em execução.

4.4 LIMITAÇÕES

A versão final do sistema terminou com alguns detalhes que na prática não se mostraram ideais.

O esquema de administradores escolhido na prática é bastante questionável. Enquanto que realmente é bastante difícil incluir o administrador de banco de dados como real parte do sistema, o fato de essa parte essencial do sistema ser totalmente externalizada e não controlada de nenhuma forma direta além das restrições impostas sobre o arquivo de configuração poderia ser melhorada. Adicionalmente, o fato de o administrador do sistema ser um ente único inalterável pode levar a trazer problemas. Poderiam ser organizados grupos de administradores e fazer com que o sistema exija um certo número de autenticações desses

administradores para realizar alguma operação como, por exemplo, alterar as credenciais de comunicação com o HSM.

O problema da configuração de banco de dados, que levou à solução apresentada em 3.2.1.1, ainda precisa ser analisado mais a fundo. A solução obtida não é perfeita, a ideia de que ela dependa de um arquivo de texto pouco intuitivo por si só já é questionável, mas também ela faz com que o sistema seja totalmente dependente de sistemas de arquivos Unix.

Voltando ao administrador do banco de dados, o fato que o mesmo indivíduo também deve ser responsável pela configuração do HTTPS no sistema é uma extrapolação um tanto estranha. Isso é necessário devido ao arquivo de configuração. O arquivo de configuração deve apenas ser acessível para o usuário do sistema operacional que executa o sistema (isso é, o administrador do banco de dados) e portanto as variáveis de ambiente também devem ser configuradas por ele. Se for possível encontrar uma solução melhor para a configuração do banco de dados, podemos separar a responsabilidade de configuração do servidor do Spring Boot do administrador do banco de dados.

Ainda, o sistema não gera logs e não possui um método de auditoria. Isso leva à necessidade de assumir que os administradores são confiáveis, como anteriormente mencionado na seção 3.2.3.

Por fim, o sistema não segue o DOC-ICP-17.01 em sua totalidade. Isso não fazia parte do escopo do trabalho, porém é importante mencionar, porque significa que o sistema não poderia ser colocado em produção na ICP-Brasil em seu estado atual. Algumas funcionalidades mencionadas no documento que não são parte do sistema incluem suporte a chaves simétricas, cifragem e autenticação de dois fatores. Além disso, o documento especifica formatos diferentes para o corpo das requisições.

5 CONCLUSÕES

O trabalho demonstra a possibilidade de desenvolvimento de um sistema web de código aberto para Prestadores de Serviço de Confiança que atende a requisitos de segurança criteriosos e cria uma sólida base sobre a qual novos sistemas podem ser desenvolvidos ou o sistema atual pode ser expandido. O código fonte do programa é disponibilizado abertamente juntamente da documentação da sua API, que está presente tanto junto do código fonte do programa como em *endpoints* do próprio sistema enquanto em execução. Dessa forma, os objetivos delineados para este trabalho podem ser considerados atingidos.

5.1 TRABALHOS FUTUROS

Para sistemas futuros, ou versões futuras do sistema, alguns detalhes que poderiam ser analisados e aperfeiçoados são o meio de configuração de banco de dados e os usuários administradores do sistema. O arquivo de configuração foi a solução utilizada, mas ela não parece ser a ideal. A configuração poderia ser feita por variáveis de ambiente, ou poderia ser encontrada uma forma de atrasar a inicialização dos objetos relacionados ao banco de dados para que a configuração possa ser feita em um *endpoint* do sistema. Quanto ao administrador, poderiam existir múltiplos, juntamente de um sistema de autenticação múltipla com pares ou grupos de administradores.

Quanto a funcionalidades do sistema, poderia ser considerada a adição de formas alternativas de armazenamento de chaves, como em arquivo ou no banco de dados, e uma forma de importar e exportar chaves do sistema. Além disso, as funções do administrador poderiam ser expandidas, para realizar ações como ativar ou desativar usuários e configurar parâmetros das chaves que o sistema utiliza para gerar *tokens* de autenticação.

REFERÊNCIAS

STALLINGS, William. **Cryptography and Network Security: Principles and Practice**. 6. ed. Prentice Hall, 2013.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. gov.br, 2021. **ICP-Brasil**. Disponível em: https://www.gov.br/iti/pt-br/assuntos/icp-brasil. Acesso em: 21 de Agosto de 2021.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. gov.br, 2021. **Lista de Prestadores de Serviço de Confiança - PSC**. Disponível em: https://www.gov.br/iti/pt-br/assuntos/icp-brasil/lista-de-prestadores-de-servico-de-confianca-psc. Acesso em: 21 de Agosto de 2021.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. **Requisitos Mínimos para as políticas de certificado na ICP-Brasil**. Thiago Meirelles Fernandes Pereira, 2020. 35 páginas. Disponível em: https://www.gov.br/iti/pt-br/assuntos/legislacao/resolucoes/Resoluo179Dec10139Etapa2DOC04_assinada.pdf. Acesso em: 15 de Agosto de 2021.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. **Procedimentos operacionais mínimos para os Prestadores de Serviço de Confiança da ICP-Brasil**. 2019. 45 páginas. Disponível em: https://www.gov.br/iti/pt-br/centrais-de-conteudo/doc-icp-17-01-verso-2-0-pdf. Acesso em: 14 de Agosto de 2021.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. **Lista de Prestadores de Serviço de Confiança**. 2021. 19 páginas. Disponível em: http://acraiz.icpbrasil.gov.br/tsl/LPSC.pdf. Acesso em: 14 de Agosto de 2021.

SPRING. **Spring Framework**., [s.d.]. Disponível em: https://spring.io/projects/spring-framework. Acesso em: 9 de Março de 2022

SPRING. **Spring Framework Documentation**., [s.d.]. Disponível em: https://spring.io/projects/spring-framework. Acesso em: 20 de Junho de 2022

MAPLE, Simon Maple. BINSTOCK, Andrew. **JVM Ecosystem report 2018 – About your Platform and Application**. 17 de Outubro de 2018. Disponível em: https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/. Acesso em: 9 de Março de 2022.

SPRINGDOC. **springdoc-openapi**., [s.d.]. Disponível em: https://springdoc.org/. Acesso em: 20 de Junho de 2022

OPENAPI INITIATIVE. **About**., [s.d.]. Disponível em: https://www.openapis.org/about. Acesso em: 20 de Junho de 2022

OPENAPI INITIATIVE. **OpenAPI Initiative Charter**., [s.d.]. Disponível em: https://www.openapis.org/participate/how-to-contribute/governance. Acesso em: 20 de Junho de 2022

OPENAPI INITIATIVE. **The OpenAPI Specification**., [s.d.]. Disponível em: https://github.com/OAI/OpenAPI-Specification/blob/main/README.md. Acesso em: 20 de Junho de 2022

AVI NETWORKS. **Hardware Security Module Definition**., [s.d.]. Disponível em: https://avinetworks.com/glossary/hardware-security-modules/. Acesso em: 26 de Junho de 2022

IBM. **Public key cryptography**. 30 de Junho de 2022. Disponível em: https://www.ibm.com/docs/en/ztpf/2022?topic=concepts-public-key-cryptography. Acesso em: 24 de Julho de 2022.

IBM. **Digital signatures**. 30 de Junho de 2022. Disponível em: https://www.ibm.com/docs/en/ztpf/2022?topic=concepts-digital-signatures. Acesso em: 24 de Julho de 2022.

KEYFACTOR. **What is PKI and How Does it Work?**, [s.d.]. Disponível em: https://www.keyfactor.com/resources/what-is-pki/. Acesso em: 27 de Julho de 2022.

HOUGEN, Aleksander. **What Is AES Encryption & How Does It Work in 2022? 256-bit vs 128-bit**, 26 de Maio de 2021. Disponível em: https://www.cloudwards.net/what-is-aes/. Acesso em: 27 de Julho de 2022.

DANIEL, Brett. **What Is AES Encryption? [The Definitive Q&A Guide]**, 21 de Março de 2021. Disponível em: https://www.trentonsystems.com/blog/aes-encryption-your-faqs-answered. Acesso em: 28 de Julho de 2022.

PAUL, Eliza. **WHAT IS DIGITAL SIGNATURE: HOW IT WORKS, BENEFITS, OBJECTIVES, CONCEPT**. 12 de Setembro de 2021. Disponível em: https://www.emptrust.com/blog/benefits-of-using-digital-signatures/. Acesso em: 30 de Julho de 2022.

ITU. **X.509: Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks**. 14 de Outubro de 2019. Disponível em: https://www.itu.int/rec/T-REC-X.509-201910-I/en. Acesso em: 30 de Julho de 2022.

IETF. **Internet Security Glossary, Version 2**. Agosto de 2007. Disponível em: https://datatracker.ietf.org/doc/html/rfc4949. Acesso em: 31 de Julho de 2022.

IETF. **Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework**. Março de 1999. Disponível em: https://datatracker.ietf.org/doc/html/rfc2527. Acesso em: 31 de Julho de 2022.

SSL.com Support Team. **What Is a Certificate Authority (CA)?**. 6 de Dezembro de 2021. Disponível em: https://www.ssl.com/faqs/what-is-a-certificate-authority/. Acesso em: 31 de Julho de 2022.

IETF. **Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1**. Fevereiro de 2003. Disponível em: https://datatracker.ietf.org/doc/html/rfc3447. Acesso em: 31 de Julho de 2022.

APÊNDICE A – Descrição do corpo das requisições e respostas do sistema

Tabela 4 – Request POST para /user

Parte do request	**Nome do campo**	**Descrição**
Body (JSON)	username	Um nome de usuário
	password	Uma senha para autenticação

Fonte: Elaboração própria

Tabela 5 – Response de POST /user

Status de resposta	**Nome do campo do JSON**	**Descrição do campo do JSON**	**Descrição da resposta**
201	username	O mesmo nome de usuário informado na criação	Indica que o usuário foi criado com sucesso
	authority	O tipo do usuário (USER ou ADMINISTRATOR).	
400	error	Uma mensagem de erro	Indica que existe um problema com as credenciais informadas ou que o sistema está desconfigurado
500	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema

Fonte: Elaboração própria

Tabela 6 – Request GET para /user

Parte do request	**Nome do campo**	**Descrição**
Header	Authorization	Bearer token de um usuário administrador

Fonte: Elaboração própria

Tabela 7 – Response de GET /user

Status de resposta	**Tipo do corpo da**	**Campo do corpo**	**Descrição do campo do JSON**	**Descrição da resposta**

	resposta			
200	Array de JSONs	username	O nome de usuário	Indica que a operação foi um sucesso e retorna os dados solicitados
		authority	O tipo do usuário (USER ou ADMINISTRATOR).	
400	JSON	error	Uma mensagem de erro	Indica que existe um problema com as credenciais informadas ou que o sistema está desconfigurado
403	JSON	error	Uma mensagem de erro	Indica que o token de autenticação não está presente ou não é válido
500	JSON	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema

Fonte: Elaboração própria

Tabela 8 – Request GET para /user/{username}

Parte do request	**Nome do campo**	**Descrição**
Header	Authorization	Bearer token de um usuário administrador.
Path	username	O nome de usuário desejado. Por exemplo, para um usuário cujo nome de usuário é “fulano”, o GET request seria enviado para /user/fulano.

Fonte: Elaboração própria

Tabela 9 – Response de GET /user

Status de resposta	**Tipo do corpo da resposta**	**Campo do corpo**	**Descrição do campo do JSON**	**Descrição da resposta**
200	JSON	username	O nome de usuário	Indica que a operação foi um sucesso e retorna os dados solicitados.
		authority	O tipo do usuário (USER ou ADMINISTRATOR).	
400	JSON	error	Uma mensagem de erro	Indica que existe um problema com as credenciais informadas ou que o sistema está desconfigurado.
403	JSON	error	Uma mensagem de erro	Indica que o token de autenticação não está presente ou não é válido.
500	JSON	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema.

Fonte: Elaboração própria

Tabela 10 – Request POST para /login

Parte do request	**Nome do campo**	**Descrição**
Body (JSON)	username	Um nome de usuário
	password	Uma senha para autenticação

Fonte: Elaboração própria

Tabela 11 – Response de GET /user

Status de resposta	**Tipo do corpo da**	**Campo do corpo**	**Descrição do campo do JSON**	**Descrição da resposta**

	resposta			
200	JSON	accessToken	O JWT para ser utilizado em endpoints que requerem autorização	Indica que a autenticação foi um sucesso e retorna o token.
401	JSON	error	Uma mensagem de erro	Indica que as credenciais não são válidas.

Fonte: Elaboração própria

Tabela 12 – Request POST para /key

Parte do request	**Nome do campo**	**Descrição**
Header	Authorization	Bearer token de um usuário comum.
Body (JSON)	keyAlgorithm	O algoritmo da chave.
	keyParamater	Parâmetro do algoritmo de criação.
	keyName	Nome que será associado ao par de chaves.

Fonte: Elaboração própria

Tabela 13 – Response de GET /user

Status de resposta	**Tipo do corpo da resposta**	**Campo do corpo**	**Descrição do campo do JSON**	**Descrição da resposta**
201	(Não há corpo)	(Não há corpo)	(Não há corpo)	Indica que a operação foi um sucesso e o par de chaves foi criado.
400	JSON	error	Uma mensagem de erro	Indica que existe um problema com os dados enviados ou que o sistema está

				desconfigurado.
403	JSON	error	Uma mensagem de erro	Indica que o token de autenticação não está presente ou não é válido.
500	JSON	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema.

Fonte: Elaboração própria

Tabela 14 – Request GET para /key

Parte do request	**Nome do campo**	**Descrição**
Header	Authorization	Bearer token de um usuário comum.
Query	keyName	Nome que foi associado ao um par de chaves em sua criação. Se esse campo não for informado, ou possuir valor vazio, o request se torna um request pelas informações de todas as chaves do usuário autenticado.

Fonte: Elaboração própria

Tabela 15 – Response de GET /key

Status de resposta	**Tipo do corpo da resposta**	**Campo do corpo**	**Descrição do campo do JSON**	**Descrição da resposta**
200	Array de JSONs	keyPairUniqueIdentifier	Identificador único do par de chaves no sistema	Indica que a operação foi um sucesso e retorna todas as chaves do usuário ou, caso o parâmetro "*keyName*" tenha sido especificado,
		keyAlgorithm	O algoritmo do par de chaves	
		publicKey	A chave pública, codificada em formato PEM	
		keyName	Nome da chave	

			especificado pelo usuário em sua criação	apenas a chave com aquele nome.
400	JSON	error	Uma mensagem de erro	Indica que existe um problema com os dados enviados ou que o sistema está desconfigurado.
403	JSON	error	Uma mensagem de erro	Indica que o token de autenticação não está presente ou não é válido.
500	JSON	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema.

Fonte: Elaboração própria

Tabela 16 – Request GET para /key/{keyUniqueIdentifier}

Parte do request	**Nome do campo**	**Descrição**
Header	Authorization	Bearer token de um usuário comum.
Path	keyUniqueIdentifier	Identificador único do par de chaves no sistema.

Fonte: Elaboração própria

Tabela 17 – Response de GET /key/{keyUniqueIdentifier}

Status de resposta	**Tipo do corpo da resposta**	**Campo do corpo**	**Descrição do campo do JSON**	**Descrição da resposta**
200	JSON	keyPairUniqueIdentifier	Identificador único do par de chaves no sistema	Indica que a operação foi um sucesso e retorna
		keyAlgorithm	O algoritmo do par de chaves	

		publicKey	A chave pública, codificada em formato PEM	as informações do par de chaves.
		keyName	Nome da chave especificado pelo usuário em sua criação	
400	JSON	error	Uma mensagem de erro	Indica que existe um problema com os dados enviados ou que o sistema está desconfigurado.
403	JSON	error	Uma mensagem de erro	Indica que o token de autenticação não está presente ou não é válido.
500	JSON	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema.

Fonte: Elaboração própria

Tabela 18 – Request POST para /key/{keyUniqueIdentifier}/sign

Parte do request	**Nome do campo**	**Descrição**
Header	Authorization	Bearer token de um usuário comum.
Path	keyUniqueIdentifier	Identificador único do par de chaves no sistema.
Body (JSON)	hashingAlgorithm (opcional)	O nome do algoritmo de hash a ser aplicado sobre os dados.
	base64EncodedData	Os dados a serem assinados codificados em Base64.

Fonte: Elaboração própria

Tabela 19 – Response de POST /key/{keyUniqueIdentifier}/sign

Status de resposta	**Tipo do corpo da resposta**	**Campo do corpo**	**Descrição do campo do JSON**	**Descrição da resposta**
200	JSON	base64EncodedSignature	A assinatura	Indica que a operação foi um sucesso e retorna a assinatura.
		keyPairUniqueIdentifier	O identificador único do par de chaves	
		base64EncodedPublicKey	A chave pública, codificada em formato PEM	
400	JSON	error	Uma mensagem de erro	Indica que existe um problema com os dados enviados ou que o sistema está desconfigurado.
403	JSON	error	Uma mensagem de erro	Indica que o token de autenticação não está presente ou não é válido.
500	JSON	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema.

Fonte: Elaboração própria

Tabela 20 – Request DELETE para /key/{keyUniqueIdentifier}

Parte do request	**Nome do campo**	**Descrição**
Header	Authorization	Bearer token de um usuário comum.
Path	keyUniqueIdentifier	Identificador único do par de chaves no sistema.

Fonte: Elaboração própria

Tabela 21 – Response de DELETE /key/{keyUniqueIdentifier}

Status de resposta	**Tipo do corpo da resposta**	**Campo do corpo**	**Descrição do campo do JSON**	**Descrição da resposta**
204	(Não há corpo)	(Não há corpo)	(Não há corpo)	Indica que a operação foi um sucesso e o par de chaves foi deletado.
400	JSON	error	Uma mensagem de erro	Indica que existe um problema com os dados enviados ou que o sistema está desconfigurado.
403	JSON	error	Uma mensagem de erro	Indica que o token de autenticação não está presente ou não é válido.
500	JSON	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema.

Fonte: Elaboração própria

Tabela 22 – Request POST para /key/{keyUniqueIdentifier}/verify-signature

Parte do request	**Nome do campo**	**Descrição**
Path	keyUniqueIdentifier	Identificador único do par de chaves no sistema.
Body (JSON)	signatureAlgorithm	O nome do algoritmo de assinatura para a verificação.
	base64EncodedData	Os dados originais.
	base64EncodedSignature	A assinatura

Fonte: Elaboração própria

Tabela 23 – Response de POST /key/{keyUniqueIdentifier}/verify-signature

Status de resposta	**Tipo do corpo da resposta**	**Campo do corpo**	**Descrição do campo do JSON**	**Descrição da resposta**
200	JSON	validSignature	Valor "*true*" caso a assinatura seja válida e "*false*" caso contrário.	Indica que a operação foi um sucesso e retorna a assinatura.
400	JSON	error	Uma mensagem de erro	Indica que existe um problema com os dados enviados ou que o sistema está desconfigurado.
500	JSON	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema.

Fonte: Elaboração própria

Tabela 24 – Request POST para /system/admin-user

Parte do request	**Nome do campo**	**Descrição**
Body (JSON)	username	Um nome de usuário
	password	Uma senha para autenticação

Fonte: Elaboração própria

Tabela 25 – Response de POST /system/admin-user

Status de resposta	**Nome do campo do JSON**	**Descrição do campo do JSON**	**Descrição da resposta**
201	username	O mesmo nome de usuário informado na criação	Indica que o usuário foi criado com sucesso
	authority	O tipo do usuário (USER ou	

		ADMINISTRATOR).	
400	error	Uma mensagem de erro	Indica que existe um problema com as credenciais informadas ou que o sistema está desconfigurado
500	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema

Fonte: Elaboração própria

Tabela 26 – Request PUT para /system/hsm-config

Parte do request	**Nome do campo**	**Descrição**
Header	Authorization	Bearer token de um usuário administrador.
Body (JSON)	parameters	Um JSON contendo os parâmetros para conexão com o HSM

Fonte: Elaboração própria

Tabela 27 – Response de PUT /system/hsm-config

Status de resposta	**Nome do campo do JSON**	**Descrição do campo do JSON**	**Descrição da resposta**
200	(Não há corpo)	(Não há corpo)	Indica que as credenciais foram registradas com sucesso
400	error	Uma mensagem de erro	Indica que existe um problema com as credenciais informadas ou que o sistema está desconfigurado
403	JSON	error	Uma mensagem de

			erro
500	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema

Fonte: Elaboração própria

Tabela 28 – Request POST para /system/hsm-config/load

Parte do request	**Nome do campo**	**Descrição**
Header	Authorization	Bearer token de um usuário administrador.

Fonte: Elaboração própria

Tabela 29 – Response de POST /system/hsm-config/load

Status de resposta	**Nome do campo do JSON**	**Descrição do campo do JSON**	**Descrição da resposta**
200	(Não há corpo)	(Não há corpo)	Indica que as credenciais foram registradas com sucesso
400	error	Uma mensagem de erro	Indica que existe um problema com as credenciais informadas ou que o sistema está desconfigurado
403	JSON	error	Uma mensagem de erro
500	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema

Fonte: Elaboração própria

Tabela 30 – Request POST para /system/refresh-keys

Parte do request	**Nome do campo**	**Descrição**

Header	Authorization	Bearer token de um usuário administrador.

Fonte: Elaboração própria

Tabela 31 – Response de POST /system/refresh-keys

Status de resposta	**Nome do campo do JSON**	**Descrição do campo do JSON**	**Descrição da resposta**
200	(Não há corpo)	(Não há corpo)	Indica que as credenciais foram registradas com sucesso
400	error	Uma mensagem de erro	Indica que existe um problema com as credenciais informadas ou que o sistema está desconfigurado
403	JSON	error	Uma mensagem de erro
500	error	Uma mensagem de erro	Indica que aconteceu algum erro interno no sistema

Fonte: Elaboração própria

APÊNDICE B – Código Fonte

O código fonte está disponível nos seguintes repositórios:

- https://github.com/guilhermesartori/TCC_PSC
- https://archive.org/details/tcc-psc-source
- https://web.archive.org/web/20220803213440/https://github.com/guilhermesartori/TCC_PSC

APÊNDICE C – Artigo SBC

Software de Código Aberto para Prestadores de Serviço de Confiança

Guilherme de Moraes Sartori

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

guimsartori@gmail.com

Abstract. *A Trust Service Provider (TSP) is an entity responsible for providing services of private key management and generation of digital signatures. TSPs are audited and inspected in Brazil by the Instituto Nacional de Tecnologia da Informação (ITI). TSPs act to facilitate the management of private keys for end users, following strict security requirements to keep essential security properties. Despite being an important part of ICP-Brasil, there is no open source software that offers all of the required features for the operation of a TSP. This paper deals with the development of an open source web system that provides this service.*

Resumo. *Um Prestador de Serviço de Confiança (PSC) é uma entidade responsável por prover serviços de gerenciamento de chaves privadas e geração de assinaturas digitais. PSCs são auditados e fiscalizados no Brasil pelo Instituto Nacional de Tecnologia da Informação (ITI). PSCs atuam para facilitar o gerenciamento de chaves privadas para os usuários finais, e fazem isso seguindo rígidos requisitos de segurança para manter propriedades de segurança essenciais. Apesar de serem uma importante parte da ICP-Brasil, não existem software aberto que oferece as funcionalidades necessárias para o funcionamento de um PSC. Este trabalho trata do desenvolvimento de um sistema web de código aberto que provê esse serviço.*

1. Introdução

A criptografia de chave pública, ou criptografia assimétrica, é um esquema de criptografia que envolve um par de chaves, uma chave pública e uma chave privada. Cada chave pública é distribuída e a sua correspondente chave privada é mantida secreta. Dados que são encriptados com a chave pública somente podem ser decriptados com a chave privada e vice-versa [IBM 2022].

Para a comunicação utilizando esse esquema de criptografia, o mensageiro encripta a mensagem com a chave pública do destinatário, porque dessa forma apenas o destinatário será capaz de decriptar a mensagem. Realizando o processo com a chave privada, é criado o que é conhecido como uma assinatura digital [IBM 2022].

A assinatura digital é diferente da cifragem com chave pública para comunicação encriptada. A assinatura digital é um dado que só pode ser gerado pela chave privada, mas que pode ser verificada por qualquer um através da correspondente chave pública. Isso faz com que, assumindo que a segurança da chave privada não foi violada, garantidamente a assinatura digital foi gerada pelo proprietário do par de chaves.

Infaestruture de Chaves Públicas, ou Public Key Infrastructure (PKI), é o conjunto de políticas, processos e papéis utilizados para gerenciamento de chaves públicas através de certificados digitais. PKI surgiu na década de 1990 para ajudar na governância de chaves criptográficas através da emissão e do gerenciamento de certificados digitais. Esses certificados são utilizados para verificar a posse de um certo par de chaves [Keyfactor, 2022].

Certificados digitais são emitidos e digitalmente assinados por Autoridades Certificadoras (AC). Essas ACs por sua vez tem seus certificados emitidos por outras ACs, criando uma cadeia de certificação que inicia em uma AC raiz com um certificado autoassinado [Keyfactor 2022]. Se uma entidade confia na AC raiz, então ela também confiará nos certificados de ACs que tenham sido emitidos pela AC raiz e também em certificados emitidos por essas ACs.

A Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil) mantém uma cadeia hierárquica de confiança para emissão e verificação de certificados digitais [ITI 2021]. As ACs raiz brasileiras assinam e emitem certificados para ACs intermediárias, que assinam e emitem certificados para ACs finais, que, finalmente assinam e emitem certificados para usuários finais.

De posse de um par de chaves, um usuário final pode obter um certificado a partir de uma AC final e realizar assinaturas que poderão ser verificadas através da chave pública de seu certificado, que tem sua validade assegurada pela assinatura da AC final e pelo resto da cadeia de certificação brasileira.

Além dos certificados para verificar a posse de um par de chaves, também é necessária a segurança do próprio par de chaves, particularmente da chave privada. Prestadores de Serviço de Confiança (PSCs) provêm esse serviço.

PSCs, dentro do contexto da ICP-Brasil, provêm armazenamento de chaves privadas para usuários finais, geração e verificação de assinaturas. PSCs são auditados e fiscalizados pelo Instituto Nacional de Tecnologia da Informação (ITI) e devem atendar às normativas estabelecidos pela ICP-Brasil [ITI 2019].

O DOC-ICP-17.01 determina os vários requisitos de segurança de PSCs que operam na ICP-Brasil, incluindo especificações de segurança pessoal, segurança física, segurança lógica, requisitos de armazenamento de chaves, entre outros. Esses são requisitos chave para garantir o pleno funcionamento, a segurança e a capacidade de auditoria de PSCs.

O ITI mantém a Lista de Prestadores de Serviço de Confiança da ICP-Brasil. A LPSC [ITI 2021] registra os PSCs credenciados para oferecer serviços à ICP Brasil e apresenta no presente momento sete entidades credenciadas.

Apesar de ser um componente importante da ICP-Brasil, não existe um software aberto ou livre disponível que realiza as funções necessárias para um PSC. Um sistema desse tipo poderia facilitar no surgimento de novas PSCs na ICP-Brasil ou poderia até mesmo substituir sistemas existentes.

A contribuição deste trabalho é, portanto, com o desenvolvimento de um software para PSCs de código aberto que siga padrões de segurança razoáveis que possa servir como ponto de partida para um eventual sistema completo que possa ser colocado em produção na ICP-Brasil.

1.1. Objetivos

O objetivo geral do trabalho é o desenvolvimento de um sistema web de código aberto que implemente funcionalidades básicas de Prestadores de Serviço de Confiança de forma razoavelmente segura. O trabalho busca criar uma base sólida que possa ser expandida para um eventual software completo ou ser utilizada como referência para um futuro software completo.

O trabalho se propõe ao desenvolvimento e entrega do código fonte de um sistema web para PSCs e o desenvolvimento da documentação da API do sistema web desenvolvido. O sistema desenvolvido dever permitir o armazenamento seguro de pares de chaves e a geração de assinaturas digitais a partir das chaves e a verificação dessas assinaturas digitais. As chaves privadas no sistema devem apenas ser utilizáveis para gerar assinaturas pelos seus proprietários

1.2. Escopo

O desenvolvimento se limita à implementação de um sistema web que forneça as funcionalidades de armazenamento de pares de chaves assimétricas e assinatura digital. Um sistema de auditoria não faz parte do escopo deste trabalho, mas a implementação do sistema busca ser razoavelmente segura dentro do limite do possível. A implementação do sistema realizada neste trabalho não teve como objetivo seguir as especificações do DOC-ICP-17.01 e, portanto, a versão final do sistema desenvolvido neste trabalho não busca ser um sistema já preparado para ser colocado em produção na ICP-Brasil.

2. Tecnologias

O sistema web foi desenvolvido na linguagem Java utilizando o framework Spring, banco de dados MySQL e arquitetura REST. A documentação do sistema web foi gerada no formato OpenAPI 3.0. O projeto do sistema foi criado utilizando o Apache Maven para gerenciamento de dependências. O sistema foi executado e testado exclusivamente no sistema operacional Ubuntu 20.04.4 LTS. Foi utilizado um Java Development Kit versão 11, como especificado no arquivo “pom.xml” do projeto.

Nas subseções a seguir, serão apresentadas e justificadas algumas das tecnologias utilizadas para o desenvolvimento do projeto.

2.1. A Linguagem Java e o Framework Spring

A linguagem Java foi escolhida para o desenvolvimento do sistema por dois motivos. Primeiro, a familiaridade e experiência do autor deste trabalho com a sua utilização. Segundo, a enorme quantidade de suporte disponível para desenvolvimento web na linguagem, tanto na biblioteca padrão quanto em diversos frameworks e bibliotecas externas.

O framework selecionado para o desenvolvimento do sistema foi o framework Spring. O Spring é um framework desenvolvido para prover um modelo compreensivo de programação e configuração para aplicações Java modernas, com extenso suporte para aplicações web. Atualmente, o Spring é o framework Java mais utilizado no mundo de acordo com o Snyk.

O Spring provê funcionalidades como injeção de dependências, mocking e gerenciamento de contextos para testes, fáceis mecanismos de acesso a banco de dados e um framework web completo que permite a separação entre aspectos de lógica de

aplicação, tratamento de respostas HTTP, filtros sobre requisições e um sistema de autenticação e autorização. Todas essas funcionalidades são extremamente úteis ou necessárias para o desenvolvimento da aplicação objetivo, o que faz do Spring uma escolha ideal.

2.2. OpenAPI e o springdoc-openapi

A OpenAPI Specification é uma especificação para interfaces de serviços web RESTful originalmente baseada na Swagger Specification. Ela é uma especificação aberta dentro da OpenAPI Initiative dirigida pela comunidade que define um padrão agnóstico a linguagens de programação de fácil interpretação por tanto humanos e computadores.

Por ser uma forma de especificação livre e amplamente utilizada, ela foi escolhida como o formato para produção da documentação da API do sistema desenvolvido.

O springdoc-openapi é uma biblioteca que ajuda na automatização da geração da documentação da API de sistemas web Spring. Essa biblioteca examina uma aplicação em execução para inferir semânticas da API baseando-se em configurações do Spring e nas definições de classes e métodos.

Por si só, o springdoc-openapi realiza uma grande porção do trabalho, mas ele não é capaz de obter informações sobre certas partes do sistema, como, por exemplo, o esquema de autenticação e autorização e, no caso da implementação atual do sistema, o formato do corpo de resposta enviado pelo servidor, que é construído dinamicamente. No entanto, isso não é um problema, porque a biblioteca também permite o uso de configurações Spring e de anotações sobre a métodos e classes para complementar a interpretação da API.

Com essa ferramenta, foi possível produzir uma documentação completa da API do sistema sem grandes dificuldades. No caso de que uma possível versão do sistema leve a alterações na API, a atualização da documentação fazendo uso do springdoc-openapi é algo trivial.

3. Análise do problema

Inicialmente foram estabelecidos os requisitos do sistema. As principais operações identificadas são a criação de chaves e realização de assinaturas com essas chaves. Para isso, seria necessário um sistema que restrinja o acesso às chaves aos seus criadores. Além disso, as chaves precisam ser mantidas de forma segura.

Para implementar essas funcionalidades básicas e seus requisitos, durante o desenvolvimento do sistema, algumas decisões precisaram ser tomadas com relação a detalhes importantes do seu funcionamento, principalmente detalhes que dizem respeito a segurança.

3.1. Usuários do sistema

Foi identificada a necessidade de três tipos de usuários para o sistema, dois tipos usuários internos do sistema e um usuário externo ao sistema: usuários comuns, o administrador do sistema e o administrador do banco de dados.

Usuários comuns são aqueles que podem utilizar o sistema para gerenciar chaves e realizar assinaturas. Eles possuem acesso aos endpoints relacionados a chaves, que os administradores não possuem.

A forma de autenticação escolhida para os usuários foi a combinação de nome de usuário e uma senha conhecida apenas pelo próprio usuário devido à sua efetividade e facilidade de aplicação. Autenticação utilizando biometria foi considerada, mas descartada devido à possível complexidade envolvida para a implementação neste trabalho. Autenticação utilizando assinatura digital também foi considerada, mas não foi ultimamente implementada por causa do esquema de armazenamento seguro desenvolvido descrito mais adiante, que é uma solução que depende de uma senha para o usuário.

Um requisito de sistema julgado importante foi a separação do papel de administrador do banco de dados e de administrador do sistema. O administrador do banco de dados gerencia exclusivamente o banco de dados do sistema enquanto a configuração de outras partes do sistema, como, em especial, as credenciais para comunicação com o HSM, são responsabilidades do administrador do sistema. Isso tem como objetivo impossibilitar que o administrador do sistema seja capaz de associar pares de chaves a seus usuários, uma vez que ele não terá acesso ao banco de dados que faz essa associação, dessa forma não sendo capaz de realizar um ataque a um usuário específico. Ao mesmo tempo isso impede que o administrador do banco de dados tenha qualquer tipo de acesso às chaves, uma vez que ele não conhece as credenciais do HSM.

A intenção inicial era que a configuração do banco de dados fosse realizada pelo administrador do banco de dados em um endpoint do sistema que armazenaria as credenciais do banco criptografadas com uma chave gerada a partir de uma senha informada pelo administrador. Porém, uma implementação desse tipo, com configuração do banco de dados em tempo de execução, se mostrou difícil devido a limitações do Spring. Esse problema foi encontrado tardiamente durante o desenvolvimento, então, em vez de uma mudança brusca como uma completa troca de framework, uma nova solução foi buscada.

A solução encontrada foi armazenar a configuração do banco de dados em um arquivo de texto que é lido pelo sistema durante a sua inicialização. O arquivo deve possuir o nome “settings.json” e ser inserido no diretório /etc/psc/. Esse arquivo deve apenas possuir permissões de leitura e escrita para o usuário responsável pelo sistema. Por fim, a pessoa que possui acesso ao usuário do sistema operacional que pode configurar o arquivo e executar o sistema deve ser o administrador do banco de dados. O sistema analisará as permissões do arquivo e rejeitará arquivos com permissões de leitura ou escrita diferentes.

Como consequência dessa decisão, apesar de o papel de administrador do banco de dados ser real e necessária para o funcionamento do sistema, esse tipo de usuário não é tratado internamente pelo sistema, uma vez que ele nunca se comunica com o sistema através de qualquer endpoint, ele é um usuário totalmente externo ao sistema.

3.2. Armazenamento seguro no banco de dados

Algumas informações precisam ser armazenadas de forma segura no banco de dados. Não apenas a senha de autenticação dos usuários, mas também os identificadores das chaves no HSM e as próprias credenciais do HSM.

Para as senhas, como em outros sistemas que fazem autenticação utilizando senhas, apenas um hash da senha é armazenado. A autenticação, mesmo com apenas um hash da senha é possível porque o sistema pode gerar o hash da senha informada pelo usuário e compará-la com o hash que possui armazenado. O algoritmo de hash escolhido

foi o bcrypt, que é o algoritmo padrão utilizado no OpenBSD, que possui uma implementação em Java facilmente integrável com o sistema de autenticação do Spring. O bcrypt requer um valor para ser utilizado como salt, protegendo o sistema de formas comuns de ataque sobre algoritmos de hashing.

Quanto às outras informações, o acesso deve ser exclusivo do usuário às quais o dado é associado. Para que isso fosse possível, a forma encontrada foi encriptar os dados com alguma informação de exclusivo conhecimento do usuário, isto é, a própria senha do usuário.

Porém, buscando evitar possíveis ataques critanalíticos que possam resultar da escolha de uma senha ruim pelo usuário, uma transformação foi escolhida para a senha. Em vez de a senha ser utilizada como chave diretamente, a chave criptográfica para é a concatenação do nome do usuário com a sua senha, e o resultado dessa concatenação novamente concatenado com a sequência “PSC”. Dessa forma, a chave é um valor que só é derivável a partir de uma informação que apenas o usuário conhece, a sua senha, e a sua segurança não depende da qualidade da senha escolhida pelo usuário.

O algoritmo simétrico de criptografia escolhido foi o AES, que é o protocolo de criptografia padrão da indústria e inquebrável através de ataques de força bruta com a tecnologia atual [Hougen 2021]. Esse algoritmo não aceita chaves de qualquer tamanho, portanto o tamanho do resultado da concatenação é ajustado para se adequar ao algoritmo. Bits são removidos ou zeros são adicionados até que o tamanho da chave se torne 256 bits.

3.3. Armazenamento de Chaves

As chaves em si são armazenadas em um HSM cujas credenciais são configuradas pelo usuário administrador do sistema. Porém, o sistema precisa armazenar tanto as credenciais de comunicação com o HSM quanto identificadores das chaves no HSM para poder acessá-las.

As credenciais do HSM que são configuradas pelo administrador são armazenadas no banco de dados criptografadas utilizando a chave descrita na seção anterior. Dentre os dados armazenados no banco dados, o identificador único da chave privada do par de chaves é armazenado criptografado com o mesmo método.

Esse esquema, juntamente da separação de papéis entre administrador do sistema e administrador do banco de dados, faz com que não seja possível que o gerenciador do banco de dados obtenha acesso às credenciais do HSM a partir das credenciais do banco de dados. O esquema de criptografia utilizado sobre o identificador da chave privada é uma camada adicional de segurança para impedir que o administrador do sistema, que possui conhecimento das credenciais do HSM, possa diretamente associar um usuário com suas chaves caso venha a obter acesso às informações do banco de dados de alguma forma ou que o administrador do banco de dados possa fazer o mesmo caso venha a obter acesso ao HSM de alguma forma.

No entanto, essa solução não é perfeita porque assume que os dois administradores não podem trabalhar juntos para comprometer as chaves dos usuários. É razoável assumir que o administrador do sistema deve ser confiável, porque é ele quem possui acesso direto às chaves privadas dos usuários no HSM. Com essa assunção, a solução aparenta ser consideravelmente segura, mas um sistema de auditoria bem implementado poderia remover a necessidade de tal assunção.

3.4. Autenticação e autorização

Evidentemente, seguindo a descrição sobre a forma escolhida para proteger informações sensíveis, o sistema precisa fazer uso de uma combinação de nome de usuário e senha para autenticação. Isso deve ser aplicado para todas as operações no sistema que envolvem utilizar ou criar uma chave. Porém, isso é muito inconveniente, principalmente para múltiplas requisições em sequência, como, por exemplo, criar um par de chaves e logo em seguida assinar dados. Uma alternativa para a autenticação que tem como objetivo evitar o constante reenvio de credenciais é o uso de um token de acesso. O usuário se autenticaria num endpoint de autenticação e obteria o token que possuiria dentro de si informações como o tipo do usuário (usuário comum ou administrador) para ser usado pelos filtros de autorização do sistema.

No entanto, à primeira vista, isso não é possível. Voltando à descrição anterior, o sistema utiliza as credenciais do usuário, incluindo sua senha, para realizar operações no banco de dados. Uma vez que todas as operações dependem das credenciais do usuário para encriptar ou decriptar alguma informação sobre chaves, o usuário precisaria se autenticar em qualquer chamada feita para o sistema, não só inviabilizando a possibilidade de um esquema de autenticação com um token de acesso, mas também tornando o uso do sistema bastante inconveniente.

A solução encontrada para esse problema é inserir essas credenciais encriptadas no próprio token utilizando uma chave que é de conhecimento exclusivo do sistema. Quando o sistema é inicializado, essa chave é gerada e ela é utilizada enquanto o sistema está em execução para assinar os tokens e para encriptar e decriptar as credenciais do usuário que são inseridas nos tokens. O formato da chave escolhido foi uma chave AES de 256 bits. A chave do sistema é gerada novamente quando o sistema é reinicializado, mas ela também pode ser atualizada com uma solicitação do administrador do sistema.

Dessa forma, o sistema é capaz fazer uso de um esquema de autenticação baseado tokens, para evitar a inconveniência de exigir as credenciais do usuário em todas as operações, e ao mesmo tempo manter o esquema de criptografia que protege as informações do banco de dados.

O sistema, então, quando recebe uma solicitação de autenticação, concatena o nome do usuário, sua senha e a string “PSC”, encripta os bytes do resultado dessa concatenação utilizando a sua chave interna via AES e converte o resultado para Base64. O resultado final é conhecido internamente no sistema como “access key” ou “chave de acesso”. Apesar de apenas um hash da senha do usuário ser mantido no banco de dados, a geração da chave de acesso é possível porque é necessário que o usuário informe sua senha durante a autenticação.

Para fazer uso da chave de acesso, o sistema decodifica o Base64, decripta essa decodificação utilizando a chave do sistema e em seguida realiza a operação desejada. Isso ocorre quando é necessário registrar informações criptografadas no banco de dados, ou obter informações criptografadas armazenadas no banco de dados, como, por exemplo, durante o processo de criação de chaves, em que a chave de acesso é utilizada para encriptar o identificador da chave primária. Esses processos de criptografia envolvendo a chave de acesso utilizam o AES.

O token de acesso utilizado emitido para o usuário quando ele se autentica no sistema é um JSON Web Token (JWT). JWTs foram definidos no RFC 7519 e são formas compactas para transportar alegações (“claims”) entre duas partes comunicantes.

JWTs podem ser assinados digitalmente para garantir sua integridade. JWTs são o formato ideal de token para o sistema porque podem carregar a chave de acesso em suas claims e podem ser assinados digitalmente pelo sistema para garantir a integridade dessa chave de acesso. Adicionalmente, JWTs também suportam uma data de expiração, o que é útil porque não é desejável que um token emitido uma única vez seja permanentemente aceitável pelo sistema, é preciso que o usuário se autentique de novo periodicamente para assegurar que ele é realmente aquele quem está acessando o sistema.

O JWT emitido pelo sistema para o usuário durante o processo de autenticação, possui duas claims: “accessKey”, que armazena a chave de acesso do usuário, e “roles”, que armazena o tipo do usuário (administrador ou usuário comum). O JWT possui um tempo de expiração e será rejeitado pelo sistema 8 horas após a sua emissão, necessitando que o usuário se autentique novamente.

O JWT é assinado com a chave do sistema utilizando HMAC-SHA256. No caso de atualização da chave do sistema, seja por reinicialização do sistema ou por requisição do administrador, JWTs assinados com a chave anterior serão rejeitados pelo sistema mesmo que não tenham atingido seu prazo de expiração, uma vez que a chave que foi utilizada para assiná-los foi perdida, o que não é grande problema porque o usuário pode simplesmente se autenticar novamente e obter um novo JWT.

De posse do JWT, o usuário pode fazer requisições sobre recursos que possuem mecanismo de autorização, como os recursos relacionados a pares de chaves no caso de usuários comuns ou recursos de configuração do sistema no caso do usuário administrador.

4. Desenvolvimento do Sistema

Com os requisitos do sistema delineados, um esqueleto da API do sistema web foi construído. A partir desse esqueleto, foi implementada um protótipo parcial do sistema. Esse protótipo foi testado através de testes de unidade e testes de integração automatizados implementados na linguagem de programação escolhida e através de testes manuais sobre uma implantação local do sistema utilizando ferramentas como o Postman para enviar requisições HTTP.

A partir do protótipo inicial, o desenvolvimento se seguiu através de ciclos em que a API do sistema era atualizada, o protótipo era refinado, recebendo funcionalidades adicionais, e os testes eram ampliados. Esse processo cíclico ocorreu paralelamente à análise descrita na seção 3, de forma que o sistema evoluiu lentamente de um protótipo inicial simples com chaves geradas em software e sem qualquer tipo de esquema de autenticação até a sua forma atual.

5. Resultados

Nesta seção, o sistema que foi desenvolvido no trabalho será apresentado em detalhes. A figura 1 mostra as relações entre as partes do sistema de uma forma simplificada.

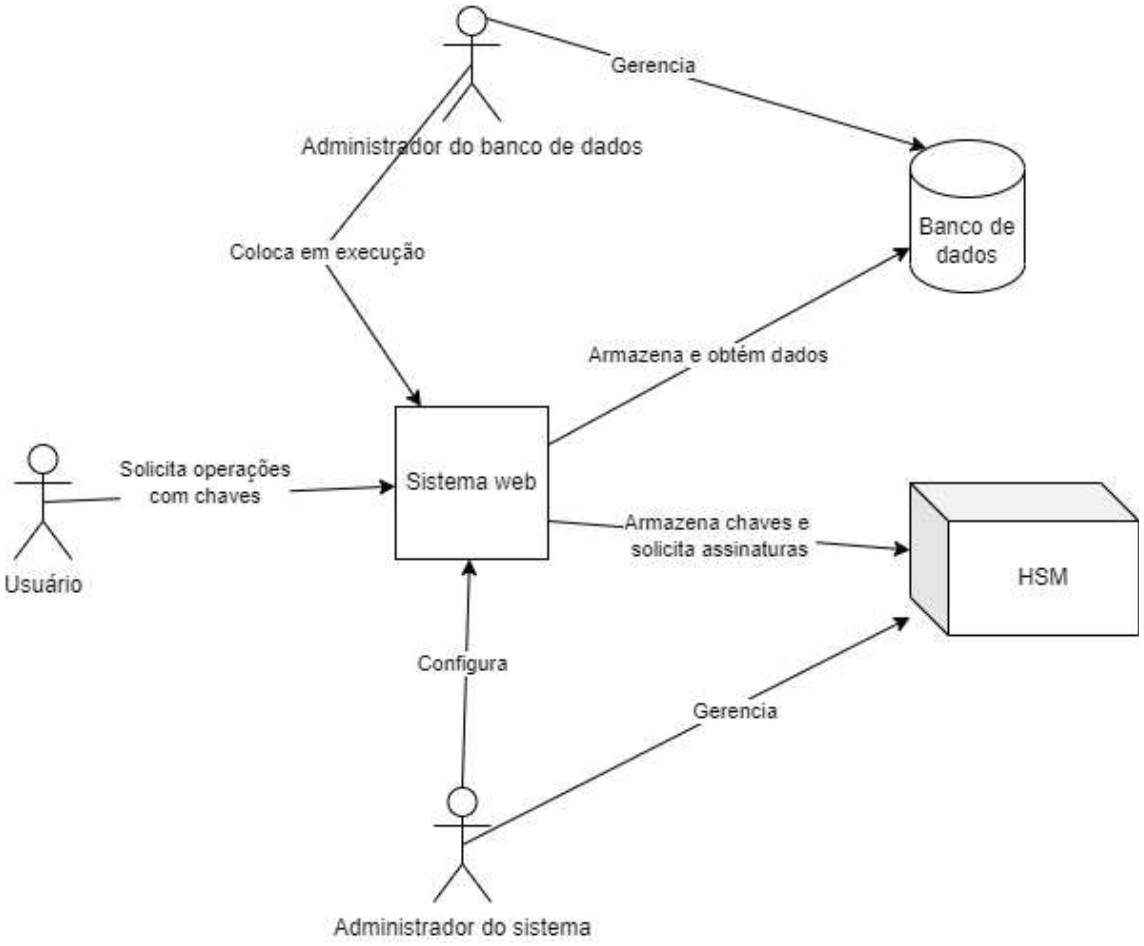


Figura 1. Diagrama do sistema

No sistema em execução, a API do sistema pode ser obtida em /v3/api-docs. A documentação também está disponível no repositório do GitHub em https://github.com/guilhermesartori/TCC_PSC/blob/master/api-docs.json. A documentação encontrada neste arquivo é padrão OpenAPI 3.0, na forma de um JSON, gerada pela biblioteca springdoc-openapi, como descrito na seção 3.1.2.

Um outro endpoint que apresenta a documentação do sistema é /swagger-ui.html. Esse endpoint apresenta uma interpretação da documentação OpenAPI em uma página web que também pode ser utilizada para enviar chamadas de teste para os recursos do sistema. O formato da documentação nessa página web é fácil de entender para aqueles que não conhecem o formato OpenAPI.

Adicionalmente, como o formato OpenAPI é amplamente suportado, várias ferramentas existem que podem fazer a interpretação da documentação para mais fácil entendimento, como o Postman, o ApiBldr, o Apicurio, plugins em IDEs como o IntelliJ IDEA, entre outros.

A tabela 1 abaixo apresenta os endpoints do sistema web, os métodos HTTP suportados por cada um e uma sucinta descrição (com exceção dos endpoints citados acima para acessos à documentação da API).

Tabela 1 – Descrição dos recursos da API do sistema

Endpoint	Métodos HTTP	Descrição
/user	POST, GET	Operações relacionadas a usuários
/user/{username}	GET	Operações relacionadas a um usuário específico. {username} é um placeholder para o nome do usuário especificado.
/login	POST	Operações de autenticação
/key	POST, GET	Operações relacionadas a pares de chaves
/key/{keyUniqueIdentifier}	GET, DELETE	Operações relacionadas a um par de chaves específico. {keyUniqueIdentifier} é um placeholder para o identificador único do par de chaves especificado.
/key/{keyUniqueIdentifier}/sign	POST	Operações relacionadas a geração de assinaturas
/key/{keyUniqueIdentifier}/verify-signature	POST	Operações relacionadas à verificação de assinaturas.
/system/admin-user	POST	Operação relacionada ao usuário administrador do sistema.
/system/hsm-config	PUT	Operação relacionada às credenciais do HSM utilizado pelo sistema.
/system/hsm-config/load	POST	Operação para carregamento de credenciais de HSM já inseridas no sistema.
/system/refresh-keys	POST	Operação para atualizar as chaves internas do sistema.

A autorização é realizada no sistema através de um filtro de requisições do framework Spring. Requisições enviadas para recursos que requerem autorização devem incluir um cabeçalho HTTP Authorization contendo a sequência de caracteres “Bearer” seguida de um espaço e o JWT emitido para o usuário no recurso de login. O JWT armazena as autoridades do usuário, isto é, se ele é um usuário comum ou o administrador, e, através da decodificação do JWT e análise dessas autoridades, requisições são filtradas ou permitidas.

A tabela 2 demonstra os requisitos de autorização para os recursos e métodos HTTP que foram configurados no filtro de requisições do Spring.

Tabela 2 – Autorização nos recursos do sistema

Recurso	Método HTTP	Autorização
/user	GET	ADMINISTRATOR
/user/{username}	GET	ADMINISTRATOR
/key	POST	USER
/key/{uniqueIdentifier}	DELETE	USER
/key/{uniqueIdentifier}	GET	USER
/key/sign	POST	USER
/system/knet-config	PUT	ADMINISTRATOR
/system/refresh-key	POST	ADMINISTRATOR

5.1. Funcionalidades

A criação de usuários ocorre através dos endpoints /user e /system/admin-user que criam

usuários comuns e administradores respectivamente através do recebimento de um nome de usuário e senha. /user também suporte obtenção de informações de usuários para princípios de administração. Usuários podem se autenticar em /login para obter o token de acesso para autorização em outros endpoints.

As operações relacionadas a chaves ocorrem nos endpoints /key. Esses endpoints podem ser usados para criar chaves, deletar chaves, obter informações de chaves, assinar com chaves e verificar assinaturas. Apenas usuários comuns podem fazer uso desses endpoints. Além disso, apenas o proprietário de um par de chaves pode deletá-lo, utilizá-lo para assinatura ou requisitar informações sobre ele. Para a criação de chaves, os algoritmos suportados são Elliptic Curve Digital Signature Algorithm (ECDSA), Rivest–Shamir–Adleman (RSA) e Edwards-curve Digital Signature Algorithm (EdDSA). Com relação a EdDSA, apenas Ed448 e Ed25519 são suportados.

Para as assinaturas que são retornadas no corpo de resposta do endpoint de assinatura, foi adotado o padrão de assinatura definido no PKCS1 (RFC 3447) devido à sua simplicidade e aplicabilidade. Este padrão é o suficiente para uma demonstração e prova de conceito, além de ser a base para outros padrões como o PKCS7. No endpoint de assinatura, o sistema envia no corpo da resposta a chave pública e o identificador do par de chaves no sistema, de forma a permitir que a verificação de assinatura possa ser feita tanto usando o endpoint de verificação de assinatura junto do identificador único do par de chaves como externamente utilizando a chave pública informada.

Os endpoints /system/ tratam da configuração do sistema e, com exceção do endpoint para criar o administrador em si, só podem ser acessados por um administrador autenticado. Ali podem ser configuradas e carregadas as credenciais de comunicação com o HSM e também pode ser atualizada a chave interna do sistema.

5.2. Limitações

A versão final do sistema terminou com alguns detalhes que na prática não se mostraram ideais.

O esquema de administradores escolhido na prática é bastante questionável. Enquanto que realmente é bastante difícil incluir o administrador de banco de dados como real parte do sistema, o fato de essa parte essencial do sistema ser totalmente externalizada e não controlada de nenhuma forma direta além das restrições impostas sobre o arquivo de configuração poderia ser melhorada. Adicionalmente, o fato de o administrador do sistema ser um ente único inalterável pode levar a trazer problemas. Poderiam ser organizados grupos de administradores e fazer com que o sistema exija um certo número de autenticações desses administradores para realizar alguma operação como, por exemplo, alterar as credenciais de comunicação com o HSM.

A forma de configuração do banco de dados ainda precisa ser analisado mais a fundo. A solução obtida não é perfeita, a ideia de que ela dependa de um arquivo de texto pouco intuitivo por si só já é questionável, mas também ela faz com que o sistema seja totalmente dependente de sistemas de arquivos Unix.

Voltando ao administrador do banco de dados, o fato que o mesmo indivíduo também deve ser responsável pela configuração do HTTPS no sistema é uma extrapolação um tanto estranha. Isso é necessário devido ao arquivo de configuração. O arquivo de configuração deve apenas ser acessível para o usuário do sistema operacional que executa o sistema (isso é, o administrador do banco de dados) e portanto as variáveis de ambiente também devem ser configuradas por ele. Se for possível encontrar

uma solução melhor para a configuração do banco de dados, podemos separar a responsabilidade de configuração do servidor do Spring Boot do administrador do banco de dados.

Ainda, o sistema não gera logs e não possui um método de auditoria. Isso leva à necessidade de assumir que os administradores são confiáveis, como anteriormente mencionado.

Por fim, o sistema não segue o DOC-ICP-17.01 em sua totalidade. Isso não fazia parte do escopo do trabalho, porém é importante mencionar, porque significa que o sistema não poderia ser colocado em produção na ICP-Brasil em seu estado atual. Algumas funcionalidades mencionadas no documento que não são parte do sistema incluem suporte a chaves simétricas, cifragem e autenticação de dois fatores. Além disso, o documento especifica formatos diferentes para o corpo das requisições.

6. Conclusões

O trabalho demonstra a possibilidade de desenvolvimento de um sistema web de código aberto para Prestadores de Serviço de Confiança que atende a requisitos de segurança criteriosos e cria uma sólida base sobre a qual novos sistemas podem ser desenvolvidos ou o sistema atual pode ser expandido. O código fonte do programa é disponibilizado abertamente juntamente da documentação da sua API, que está presente tanto junto do código fonte do programa como em endpoints do próprio sistema enquanto em execução. Dessa forma, os objetivos delineados para este trabalho podem ser considerados atingidos.

Para sistemas futuros, ou versões futuras do sistema, alguns detalhes que poderiam ser analisados e aperfeiçoados são o meio de configuração de banco de dados e os usuários administradores do sistema. O arquivo de configuração foi a solução utilizada, mas ela não parece ser a ideal. A configuração poderia ser feita por variáveis de ambiente, ou poderia ser encontrada uma forma de atrasar a inicialização dos objetos relacionados ao banco de dados para que a configuração possa ser feita em um endpoint do sistema. Quanto ao administrador, poderiam existir múltiplos, juntamente de um sistema de autenticação múltipla com pares ou grupos de administradores.

Quanto a funcionalidades do sistema, poderia ser considerada a adição de formas alternativas de armazenamento de chaves, como em arquivo ou no banco de dados, e uma forma de importar e exportar chaves do sistema. Além disso, as funções do administrador poderiam ser expandidas, para realizar ações como ativar ou desativar usuários e configurar parâmetros das chaves que o sistema utiliza para gerar tokens de autenticação.

References

STALLINGS, William. Cryptography and Network Security: Principles and Practice. 6. ed. Prentice Hall, 2013.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. gov.br, 2021. ICP-Brasil. Disponível em: https://www.gov.br/iti/pt-br/assuntos/icp-brasil. Acesso em: 21 de Agosto de 2021.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. gov.br, 2021. Lista de Prestadores de Serviço de Confiança - PSC. Disponível em: https://www.gov.br/iti/pt-br/assuntos/icp-brasil/lista-de-prestadores-de-servico-de-

confianca-psc. Acesso em: 21 de Agosto de 2021.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. Requisitos Mínimos para as políticas de certificado na ICP-Brasil. Thiago Meirelles Fernandes Pereira, 2020. 35 páginas. Disponível em: https://www.gov.br/iti/pt-br/assuntos/legislacao/resolucoes/Resoluo179Dec10139Etapa2DOC04_assinada.pdf. Acesso em: 15 de Agosto de 2021.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. Procedimentos operacionais mínimos para os Prestadores de Serviço de Confiança da ICP-Brasil. 2019. 45 páginas. Disponível em: https://www.gov.br/iti/pt-br/centrais-de-conteudo/doc-icp-17-01-verso-2-0-pdf. Acesso em: 14 de Agosto de 2021.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. Lista de Prestadores de Serviço de Confiança. 2021. 19 páginas. Disponível em: http://acraiz.icpbrasil.gov.br/tsl/LPSC.pdf. Acesso em: 14 de Agosto de 2021.

SPRING. Spring Framework., [s.d.]. Disponível em: https://spring.io/projects/spring-framework. Acesso em: 9 de Março de 2022

SPRING. Spring Framework Documentation., [s.d.]. Disponível em: https://spring.io/projects/spring-framework. Acesso em: 20 de Junho de 2022

MAPLE, Simon Maple. BINSTOCK, Andrew. JVM Ecosystem report 2018 – About your Platform and Application. 17 de Outubro de 2018. Disponível em: https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/. Acesso em: 9 de Março de 2022.

SPRINGDOC. springdoc-openapi., [s.d.]. Disponível em: https://springdoc.org/. Acesso em: 20 de Junho de 2022

OPENAPI INITIATIVE. About., [s.d.]. Disponível em: https://www.openapis.org/about. Acesso em: 20 de Junho de 2022

OPENAPI INITIATIVE. OpenAPI Initiative Charter., [s.d.]. Disponível em: https://www.openapis.org/participate/how-to-contribute/governance. Acesso em: 20 de Junho de 2022

OPENAPI INITIATIVE. The OpenAPI Specification., [s.d.]. Disponível em: https://github.com/OAI/OpenAPI-Specification/blob/main/README.md. Acesso em: 20 de Junho de 2022

AVI NETWORKS. Hardware Security Module Definition., [s.d.]. Disponível em: https://avinetworks.com/glossary/hardware-security-modules/. Acesso em: 26 de Junho de 2022

IBM. Public key cryptography. 30 de Junho de 2022. Disponível em: https://www.ibm.com/docs/en/ztpf/2022?topic=concepts-public-key-cryptography. Acesso em: 24 de Julho de 2022.

IBM. Digital signatures. 30 de Junho de 2022. Disponível em: https://www.ibm.com/docs/en/ztpf/2022?topic=concepts-digital-signatures. Acesso em: 24 de Julho de 2022.

KEYFACTOR. What is PKI and How Does it Work?, [s.d.]. Disponível em: https://www.keyfactor.com/resources/what-is-pki/. Acesso em: 27 de Julho de 2022.

HOUGEN, Aleksander. What Is AES Encryption & How Does It Work in 2022? 256-bit vs 128-bit, 26 de Maio de 2021. Disponível em: https://www.cloudwards.net/what-is-aes/. Acesso em: 27 de Julho de 2022.

DANIEL, Brett. What Is AES Encryption? [The Definitive Q&A Guide], 21 de Março de 2021. Disponível em: https://www.trentonsystems.com/blog/aes-encryption-your-faqs-answered. Acesso em: 28 de Julho de 2022.

PAUL, Eliza. WHAT IS DIGITAL SIGNATURE: HOW IT WORKS, BENEFITS, OBJECTIVES, CONCEPT. 12 de Setembro de 2021. Disponível em: https://www.emptrust.com/blog/benefits-of-using-digital-signatures/. Acesso em: 30 de Julho de 2022.

ITU. X.509: Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks. 14 de Outubro de 2019. Disponível em: https://www.itu.int/rec/T-REC-X.509-201910-I/en. Acesso em: 30 de Julho de 2022.

IETF. Internet Security Glossary, Version 2. Agosto de 2007. Disponível em: https://datatracker.ietf.org/doc/html/rfc4949. Acesso em: 31 de Julho de 2022.

IETF. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. Março de 1999. Disponível em: https://datatracker.ietf.org/doc/html/rfc2527. Acesso em: 31 de Julho de 2022.

SSL.com Support Team. What Is a Certificate Authority (CA)?. 6 de Dezembro de 2021. Disponível em: https://www.ssl.com/faqs/what-is-a-certificate-authority/. Acesso em: 31 de Julho de 2022.

IETF. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. Fevereiro de 2003. Disponível em: https://datatracker.ietf.org/doc/html/rfc3447. Acesso em: 31 de Julho de 2022.