**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

**ALEXANDRE HERRERO MATIAS**

# PUBLIC KEY CHAIN: UM SISTEMA GRÁFICO DE INFRAESTRUTURA DE CHAVES PÚBLICAS NA BLOCKCHAIN

**CURITIBA**

**2022**

**ALEXANDRE HERRERO MATIAS**

# PUBLIC KEY CHAIN: UM SISTEMA GRÁFICO DE INFRAESTRUTURA DE CHAVES PÚBLICAS NA BLOCKCHAIN

## Public Key Chain: A Graphic Public Key Infrastructure System on Blockchain

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Mauro Sergio Pereira Fonseca

**CURITIBA**

**2022**

**ALEXANDRE HERRERO MATIAS**

# PUBLIC KEY CHAIN: UM SISTEMA GRÁFICO DE INFRAESTRUTURA DE CHAVES PÚBLICAS NA BLOCKCHAIN

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 08/dezembro/2022

---

Mauro Sergio Pereira Fonseca
Doutorado
Universidade Tecnológica Federal do Paraná

---

Hermes Irineu Del Monego
Doutorado
Universidade Tecnológica Federal do Paraná

---

Fabiano Scriptore de Carvalho
Mestrado
Universidade Tecnológica Federal do Paraná

**CURITIBA**

**2022**

## AGRADECIMENTOS

Infelizmente não será possível agradecer a todas as pessoas que fizeram parte da minha jornada até este momento, desta forma já peço desculpas àqueles que não se fizeram presentes nestas palavras.

Agradeço primeiramente a minha família por todo seu apoio e incentivos, não só durante minha formação acadêmica, mas como em toda minha vida. A meu pai Lucio, minha mãe Marcia, meus irmãos Lucas e Leonardo e também a meus avós Francisco e Dolores.

Ao meu orientador, o Prof. Mauro, ao qual tive o prazer de também ter feito a disciplina de Segurança de Redes e Sistemas, agradeço toda a assistência e instrução durante a realização deste projeto.

Aos meus amigos mais próximos Giuliana e Matheus, por todos os momentos, bons e ruins, que passamos juntos durante esses anos de UTFPR.

Um agradecimento especial ao Programa de Educação Tutorial Computando Cultura em Equidade, que foi uma parte essencial da minha formação tanto acadêmica como pessoal. A professora tutora Marília por nos guiar e coordenar sempre de maneira horizontal.

E também a todos que de alguma forma ou de outra contribuíram para a realização deste projeto.

「何でもは知らないわよ。知ってることだけ。」羽川翼

## RESUMO

Confidencialidade é um dos três pilares fundamentais da segurança da informação; é o que garante que certas informações possam ser acessadas apenas por pessoas autorizadas. Muitas vezes passa despercebido, porém, sua utilização se faz presente no dia a dia através do protocolo Hyper Text Transfer Protocol Secure (HTTPS). Este que, também, é uma forma de confidencialidade e garante uma navegação *web* particular e segura. Entretanto, para seu funcionamento pleno, é necessária a existência de um sistema por de trás dos panos, a Public Key Infrastructure (PKI). Este projeto consiste na implementação de uma PKI fazendo uso da tecnologia da *Blockchain* como auxilio para garantir suas propriedades de confidencialidade.



**Palavras-chave:** blockchain; infraestrutura; chave; certificado; segurança.

**ABSTRACT**

Confidentiality is one of the three fundamental pillars of information security; is what ensures that certain information can only be accessed by authorized persons. It often goes unnoticed, but its use is present in everyday life through the protocol HTTPS. Which is also a form of confidentiality and guarantees a private and safe navigation. However, for its full functioning, the existence of a behind-the-scenes system, the PKI, is necessary. This project consists in the implementation of a PKI making use of the *Blockchain* technology as an aid to guarantee its confidentiality properties.

**Keywords:** blockchain; infrastructure; key; certificate; security.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

### Siglas

| | |
|---|---|
| CA | Certificate Authority |
| CERT | Computer Emergency Response Team |
| DNS | Domain Name Service |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| EdDSA | Edwards-curve Digital Signature Algorithm |
| EVM | Ethereum Virtual Machine |
| GNU | GNU's Not Unix |
| HTTP | Hyper Text Transfer Protocol |
| HTTPS | Hyper Text Transfer Protocol Secure |
| IPFS | InterPlanetary File System |
| LGLP | GNU Lesser General Public License |
| LRA | Local Register Authority |
| MB | Mega Byte |
| PKC | Public Key Chain |
| PKI | Public Key Infrastructure |
| PoA | Proof of Authority |
| PoS | Proof of Stake |
| PoW | Proof of Work |
| PSK | Pre-Shared Key |
| RA | Register Authority |
| RSA | Rivest-Shamir-Adleman |
| SSD | Solid State Drive |

| | |
|---|---|
| TB | Tera Byte |
| TLS | Transport Layer Security |
| TTP | Trusted Third Parties |
| URL | Uniform Resource Locators |

**SUMÁRIO**

# 1 INTRODUÇÃO

Um dos pilares principais da segurança da informação é a confidencialidade. Esta é definida como a garantia de que a informação apenas se encontra disponível para quem está autorizado a acesa-la. Existem inúmeras soluções para o problema de confidencialidade em sistemas web, dentre elas a mais utilizada é o protocolo Transport Layer Security (TLS). A base do protocolo TLS é feita com a utilização de criptografia assimétrica, o qual faz uso de um par de chaves publica e privada. A chave privada é matinada sigilosa, enquanto a chave publica é distribuída. Seu comportamento garante que uma mensagem cifrada com a chave publica somente pode ser decifrada pela chave privada. Garantindo, assim, que apenas o dono da chave privada possa ler a mensagem, garantindo assim a confidencialidade. A distribuição destas chaves é feita com auxilio de uma PKI.

Contudo, esta solução apresenta alguns problemas. Como é possível confiar cegamente em uma chave pública, como é possível ter certeza que o dono desta chave realmente é quem diz que é? Uma solução encontrada para este problema foi a utilização de Certificate Authority (CA), companhias confiáveis que irão garantir para o usuário que a chave pública é verdadeira e confiável. Estas CAs, em meio a cobrança de uma taxa, fazem a assinatura de certificados digitais para os usuários.

Por mais que sistemas baseados em CAs sejam os mais utilizando atualmente para navegação *web*, não quer dizer que sejam perfeitos ou imunes a falhas. Um incidente notório com uma violação de CAs ocorreu em julho de 2011, quando a CA holandesa DigiNotar ficou sabendo que seus sistemas haviam sido hackeados e, por aproximadamente 2 meses, cobriu o incidente. Em agosto do mesmo ano, uma organização irmã do Computer Emergency Response Team (CERT) holandês informou que um usuário iraniano havia postado em um fórum público que estava recebendo mensagens de erro sobre um possível certificado fraudulento ao tentar acessar o *Google*. Com isso, descobriu-se que nesses 2 meses de obscuridade centenas de certificados falsos haviam sido criados (ARNBAK; EIJK, 2012).

## 1.1 Motivação

A utilização de CAs para validação de certificados digitais, apesar de robusta, apresenta grandes pontos fracos de segurança. O fator humano presente em qualquer CA pode ser comprometedor, as CAs podem ser comprometidas devido a violações de segurança e certificados falsos podem ser criados para que usuários mal-intencionadas se façam passar por alguma entidade. Este problema é ainda pior em CAs, que fazem uso de Trusted Third Parties (TTP), o que aumenta ainda mais a área de superfície de ataque para uma infraestrutura de segurança (BERKOWSKY; HAYAJNEH, 2017).

Com isso a motivação para o desenvolvimento deste projeto é a utilização da tecnologia da *Blockchain* como uma alternativa. A *Blockchain* é uma tecnologia em ascensão que faz o

uso muito inteligente de resumos criptográficos (funções *hash* criptográficas) e algoritmos de consenso para garantir a autenticidade, integridade e confiabilidade de seus dados (NOFER *et al.*, 2017).

CAs, além de não serem totalmente seguras, também apresentam um alto custo (podendo variar de *R$100,00* - *R$500,00* por ano dependo da tipo do certificado e da CA escolhidos) (MARQUEZ, 2018). Outra motivação é reduzir esse valor o máximo possível, cobrando do usuário apenas o mínimo necessário para utilização da *Blockchain*.

## 1.2 Proposta

A proposta deste projeto é o desenvolvimento de um sistema capaz de funcionar como uma alternativa para as CAs utilizando a *Blockchain* como sua base.

### 1.2.1 Objetivo Geral

O objetivo geral deste trabalho é o desenvolvimento de um sistema capaz de propor a utilização da *Blockchain* como uma alternativa para o modelo baseado em CAs de PKI. O sistema deve apresentar as funcionalidades básicas de uma PKI, emissão de certificados, armazenamento de certificados, distribuição de certificados, revogação de certificados e, o mais importante, garantia de autenticidade.

### 1.2.2 Objetivos Específicos

Os objetivos específicos são os seguintes:

1. Desenvolver o sistema respeitando suas especificações;
2. Adicionar uma funcionalidade de geração automática de certificados X509 e de par de chaves RSA no próprio sistema;
3. Adicionar uma funcionalidade de instalação e remoção automática de certificados na máquina bem como nos navegadores web do usuário;
4. Desenvolver o sistema com uma interface gráfica, visando adicionar o mínimo possível de complexidade para o usuário comum;
5. Desenvolver uma extensão para navegador que irá facilitar ainda mais a operação;
6. Realizar a validação do funcionamento do sistema simulando um ambiente real pela a perspectiva de um usuário comum bem como de um desenvolvedor.

# 2 REFERENCIAL TEÓRICO

Neste capítulo será apresentado o referencial teórico necessário para a concepção e desenvolvimento do projeto, a fim de atender o objetivo geral proposto.

## 2.1 *Transport Layer Security*

O objetivo principal do TLS é fornecer um canal seguro entre dois pares comunicantes; o único requisito para este transporte é um fluxo de dados confiável e ordenado. Especificamente, este canal seguro deve garantir as seguintes propriedades:

- **Autenticidade**: O lado do servidor do canal é sempre autenticado; o lado do cliente é opcionalmente autenticado. A autenticação pode acontecer através de criptografia assimétrica (por exemplo, RSA, Elliptic Curve Digital Signature Algorithm (ECDSA), Edwards-curve Digital Signature Algorithm (EdDSA)) ou uma chave simétrica pré-compartilhada (Pre-Shared Key (PSK)).
- **Confidencialidade**: Os dados enviados através do canal após o estabelecimento só são visíveis para os *endpoints*. O TLS não esconde o comprimento dos dados que transmite, embora os *endpoints* sejam capazes de preencher os registros TLS a fim de obscurecer os comprimentos e melhorar a proteção contra técnicas de análise de tráfego.
- **Integridade**: Os dados enviados pelo canal após o estabelecimento não podem ser modificados por atacantes sem serem detectados.

Estas propriedades devem ser verdadeiras mesmo diante de um atacante que tenha controle completo da rede (RESCORLA, 2018).

### 2.1.1 Par de chaves públicas/privadas

Criptografia assimétrica faz uso de pares de chaves públicas/privadas relacionadas matematicamente. Tipicamente, uma destas chaves é tornada pública, publicando-se na internet, por exemplo, enquanto a outra permanece privada. A criptografia assimétrica (ou de chave pública) funciona de tal forma que uma mensagem criptografada com a chave pública só pode ser descriptografada com sua chave privada e, inversamente, uma mensagem assinada com uma chave privada pode ser verificada com sua chave pública (American Institute of Certified Public Accountants (AICPA) and Canadian Institute of Chartered Accountants, 2000).

### 2.1.2 Certificados Digitais

Os usuários de chaves públicas exigem confiança de que a chave privada associada é realmente propriedade do sujeito remoto (pessoa ou sistema) correto com o qual será usado o mecanismo de criptografia ou assinatura digital. Esta confiança é obtida através do uso de certificados digitais de chave pública, que são estruturas de dados que ligam os valores das chaves públicas aos sujeitos. A vinculação é afirmada através de uma assinatura digital de cada certificado. Essa assinatura pode ser feita pelo próprio sujeito (certificado autoassinado) ou por uma entidade de confiança (CA). Um certificado tem uma vida útil limitada, que é indicada em seu conteúdo assinado. Como a assinatura e a pontualidade de um certificado podem ser verificadas independentemente por um usuário de certificado, os certificados podem ser distribuídos através de comunicações e sistemas não confiáveis, e podem ser armazenados em caches de sistemas não seguros (HOUSLEY *et al.*, 2002).

## 2.2 Infraestrutura de Chaves Públicas

Uma PKI é definida como a base de uma infraestrutura de segurança abrangente cujos serviços são implementados e prestados utilizando conceitos e técnicas de criptografia de chaves públicas (ADAMS; LLOYD, 2002).

Uma PKI totalmente funcional, por ser a base de uma infra-estrutura de segurança, também engloba um grande número de serviços, como, o armazenamento de certificados, a revogação de certificados, o backup e recuperação de chaves, a atualização automática de chaves, o gerenciamento do histórico de chaves, certificações cruzadas, o suporte para não repúdio, o carimbo de tempo, entre outros (ADAMS; LLOYD, 2002).

Em uma PKI é necessário existir confiança de que uma chave pública associada é pertencente a entidade correta com a qual se deseja usar um mecanismo de criptografia ou assinatura digital. Esta confiança é obtida através do uso de certificados digitais que vinculam esta chave a uma entidade. Este vinculo é afirmado através da assinatura digital destes certificados por uma CA de confiança (BOEYEN *et al.*, 2008).

### 2.2.1 Autoridades Certificadoras

A premissa básica é que uma CA atesta o vínculo entre um indivíduo e sua chave pública. A CA fornece um nível de garantia de que a chave pública contida no certificado pertence de fato à entidade indicada no mesmo. A assinatura digital colocada no certificado de chave pública pela CA fornece a ligação criptográfica entre a chave pública da entidade, o nome da entidade e outras informações no certificado, como uma data de validade (American Institute of Certified Public Accountants (AICPA) and Canadian Institute of Chartered Accountants, 2000).

Como as CAs emitem os certificados digitalmente, e estes já estão protegidos do ponto de vista da integridade, os certificados podem ser divulgados na internet livremente, assumindo que não contenham nenhuma informação sensível (ADAMS; LLOYD, 2002).

As CAs podem assumir uma série de representações diferentes, dependendo do modelo de confiança corporizado pela PKI. Por exemplo, em um ambiente corporativo, pode-se esperar que uma ou mais autoridades sejam responsáveis pela emissão de certificados para os funcionários da empresa (ADAMS; LLOYD, 2002).

### 2.2.2 Autoridades de Registro

Em alguns casos as CAs podem delegar suas funções de identificação e autenticação de assinantes para um entidade conhecida como Register Authority (RA). Nestes casos, as RAs realizam internamente a função de registro de assinantes. Em outros casos, a CA pode delegar a função de RA a outras autoridades externas (às vezes chamadas de Local Register Authority (LRA)) que podem ou não fazer parte da mesma entidade legal da CA. Essas RAs externas são obrigadas a cumprir as disposições relevantes das práticas comerciais divulgadas pela CA (American Institute of Certified Public Accountants (AICPA) and Canadian Institute of Chartered Accountants, 2000).

## 2.3 *Blockchain*

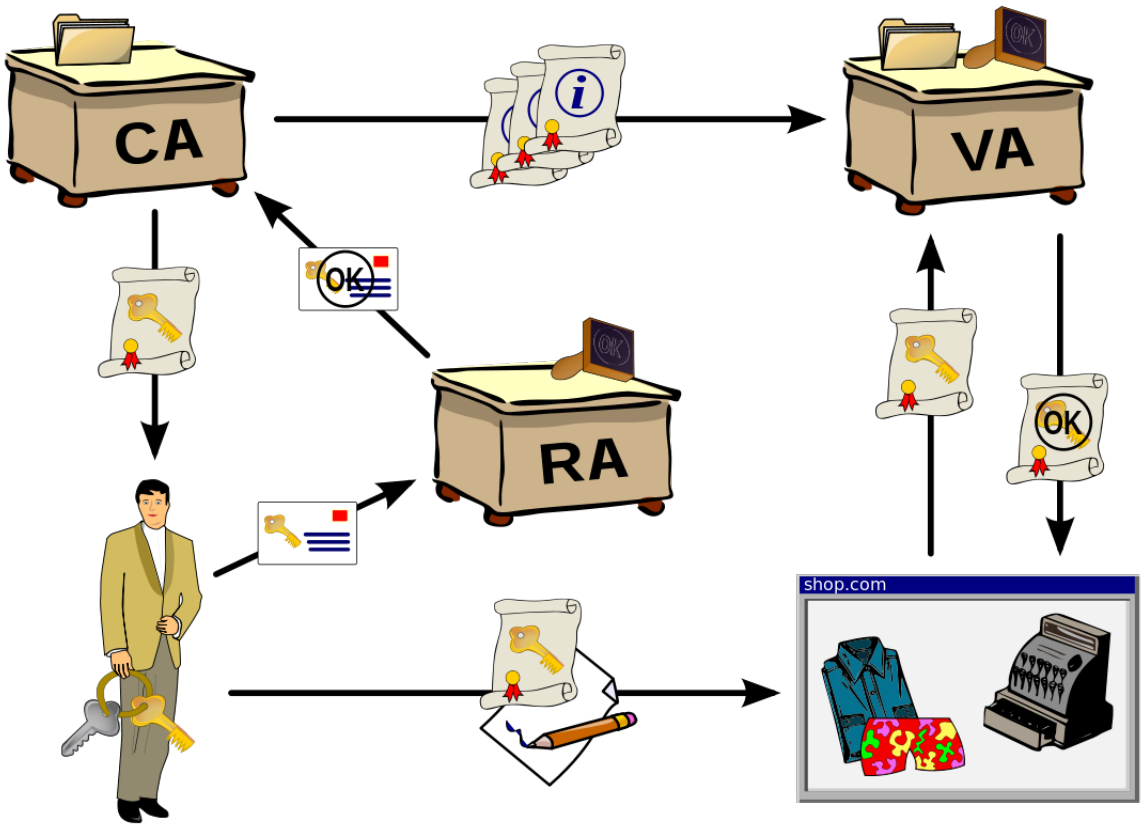
*Blockchain* é um sistema de registro distribuído que se baseia em uma lista crescente de blocos que armazenam transações. Estes blocos contem, de forma geral, uma identificação para o bloco antecessor (função *hash* do bloco anterior), uma informação de *timestamp*, um numero aleatório utilizado para verificação (*nonce*) além do histórico de transações (NAKAMOTO, 2009).

O pilar principal para o funcionamento de uma *Blockchain* são as funções *hash* que geram valores únicos e ajudam a evitar qualquer tipo de fraude, uma vez que a mudança de um bloco se quer na cadeia mudaria imediatamente o respectivo valor de sua função *hash* (NOFER *et al.*, 2017).

Contudo, também é necessário uma garantia de que os blocos que compõe a *Blockchain* são válios. Para isso é utilizado um mecanismo de consenso que irá garantir a validade dos blocos que entram (NOFER *et al.*, 2017). Os principais mecanismos de consenso utilizados são Proof of Work (PoW), Proof of Stake (PoS) e Proof of Authority (PoA).

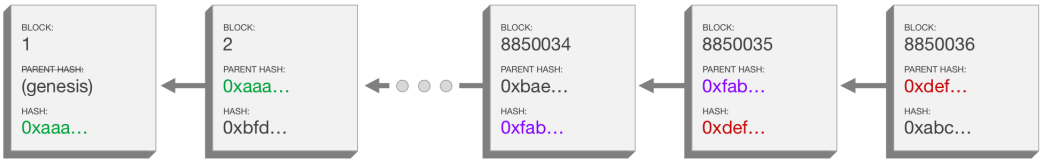
Em um mecanismo PoW, os participantes precisam adivinhar um numero aleatório ao ajustar o valor do *nonce* de seus blocos e calcular suas respectivas funções *hash*. Visto que as funções *hash* são valores pseudoaleatórios, os participantes não tem opção melhor do que testar o máximo possível de valores para o *nonce* em busca da *hash* correta. O primeiro participante a encontrar este valor é permitido adicionar o bloco a *Blockchain*. Desta forma, este

**Figura 1 – Modelo simplificado de uma Infraestrutura de Chaves Públicas**




**Fonte: Adaptado de Cris (2022).**

**Figura 2 – Estrutura dos blocos em uma *Blockchain***




**Fonte: Adaptado de Garreau (2022).**

mecanismo favorece aqueles que dedicam maior poder computacional como mais confiáveis (BUTERIN, 2015).

Já em um mecanismo PoS, a confiabilidade de um participante é medida de acordo com a quantidade de fundos (criptoativo associada a *Blockchain*) que este possui, visto que estes devem se manter honestos para não arriscar seus grande investimento. Com isso, este sistema evita o grande desperdiço energético causado em um mecanismo PoW (BUTERIN; GRIFFITH, 2017).

Por fim um mecanismo PoA, a confiabilidade dos participantes depende do seu grau de reputação. Esta reputação pode ser calculada de diferentes formas dependendo do algoritmo de consenso (ANGELIS *et al.*, 2017).

Uma vez que um bloco é validado pelo mecanismo de consenso e definitivamente adicionado a *Blockchain* este não pode ser modificado ou removido.

### 2.3.1 *Smart Contracts*

*Smart Contracts* são definidos como sistemas que aplicam regras pré-especificadas arbitrárias, sendo em sua maior parte relacionados ativos digitais (BUTERIN, 2015). Muitas *Blockchains* mais modernas hoje apresentam suporte para *Smart Contracts*.

As *Blockchain*s mais tradicionais, como o *BitCoin*, não apresentam suporte a *Smart Contracts* e, desta forma, não apresentam suporte para desenvolvimento e execução de código. Contudo, já existem muitas plataformas que suportam esta tecnologia, inclusive existem linguagens de programação desenvolvidas exclusivamente para este tipo de desenvolvimento.

## 2.4 Ethereum

Ethereum é uma *Blockchain* compatível com *Smart Contracts* que faz uso de sua máquina virtual Ethereum Virtual Machine (EVM) para fazer execução de *Smart Contracts*. Solidity e Vyper são exemplos de linguagens para o desenvolvimento de *Smart Contracts* na *Blockchain* Ethereum.

Os objetivos da Ethereum são fundir e melhorar os conceitos de *scripting*, *altcoins* e meta-protocolos em *Blockchain*, além de permitir que os desenvolvedores criem aplicações baseadas em algoritmos de consenso arbitrários que tenham a escalabilidade, padronização, completude, facilidade de desenvolvimento e interoperabilidade oferecidas por esses diferentes paradigmas (BUTERIN, 2015).

### 2.4.1 Solidity

Solidity é uma linguagem de programação orientada a objetos desenvolvida para a implementação de *Smart Contracts* na *Blockchain* Ethereum. Solidity é uma linguagem fortemente tipada e que se utiliza de chaves, muito influenciada por C++, JavaScript e Python (Solidity Team, 2022).

### 2.4.2 Vyper

Da mesma forma que Solidity, Vyper também é uma linguagem de programação orientada a objetos desenvolvida para a implementação de *Smart Contracts* na Ethereum. Vyper é uma linguagem pythônica, focada muito mais nas questões de segurança.

Vyper é nova e moderna, uma de suas vantagens sobre Solidity são as correções de varias de suas vulnerabilidades (Vyper Team, 2022).

### 2.4.3 *Framework* Brownie

Brownie é um *framework* de desenvolvimento e testes baseado em Python para *Smart Contracs* visando a plataforma Ethereum.

Dentre as funcionalidades do Brownie estão, suporte completo para Solidity e Vyper, teste de *Smart Contracts*, ferramentas de depuração incluindo *traces* no estilo Python e *strings* de erros personalizadas, console embutido para interação rápida do projeto, entre outros.

### 2.4.4 Ambiente de testes

Toda transação feita na *Blockchain* por um *Smart Contract* tem um custo associado a sua complexidade computacional. Esse custo, comumente chamado de *gas*, é pago diretamente da carteira de criptoativos do usuário que realizou a transação.

Desta forma se torna inviável a realização de testes em *Blockchains* reais, ou seja, que tem valor associado a seus criptoativos. A solução é a utilização de ambientes de testes desenvolvidos para este proposito.

Existem duas abordagem para ambientes de testes, simuladores de *Blockchain* e redes de teste.

#### 2.4.4.1 Ganache

Ganache é um simulador de *Blockchain* que permite hospedar uma *Blockchain* pessoal que pode ser usada para executar testes, executar comandos e inspecionar seu estado enquanto controla como a *Blockchain* opera (ConsenSys Software Inc, 2022).

#### 2.4.4.2 Rede de testes Rinkeby

Uma rede de testes em *Blockchain* é uma interface da *Blockchain* que funciona exatamente como sua rede principal porém seus criptoativos são distribuídos gratuitamente para os desenvolvedores que desejam testar suas aplicações antes de implementa-las em uma rede principal.

A *Blockchain* Ethereum possui 5 redes de teste, Görli, Sepolia, Ropsten, Rinkeby e Kovan. Destas a mais utilizada por muito tempo foi a rede de testes Rinkeby, visto ser mais segura a ataques de negação de serviço do que Ropsten e mais simples de se configurar do que Kovan.

Rinkeby, juntamente com Ropsten e Kovan, serão descontinuadas até o final de 2022. Desta forma os desenvolvedores foram indicados a migrarem seus projetos para as duas redes de teste que serão mantidas, Görli e Sepolia.

#### 2.4.4.3 Rede de testes Görli

A rede de testes Görli é a primeira rede de testes PoA *cross-client*. É sincronizada com Parity Ethereum, Geth, Nethermind, Hyperledger Besu (antigo Pantheon), e EthereumJS. Esta rede de testes é um projeto baseado na comunidade, completamente de código aberto.

### 2.4.5 Conexão com a *Blockchain*

Para se realizar qualquer interação com uma *Blockchain* existem duas abordagens que podem ser utilizadas, nó completo local da *Blockchain* e nós hospedados.

#### 2.4.5.1 Go Ethereum (Geth)

Go Ethereum é uma das três implementações originais (juntamente com C++ e Python) do protocolo Ethereum. Escrito em Go, totalmente de código aberto e licenciado sob a GNU's Not Unix (GNU) GNU Lesser General Public License (LGLP) v3, é utilizado para hospedar localmente um nó completo da Ethereum (The go-ethereum Authors, 2022).

#### 2.4.5.2 Infura

Infura é um serviço de nós hospedados de *Blockchain*. Fornecendo ferramentas e infraestrutura que permitem os desenvolvedores facilmente levar suas aplicações de *Blockchain*, dês dos testes à implantação em produção, com acesso simples e confiável a Ethereum e InterPlanetary File System (IPFS) (Infura Inc, 2022).

# 3 TRABALHOS RELACIONADOS

Alguns trabalhos desenvolvidos para o mesmo tópico já existem na literatura. O mais conhecido é o projeto CertCoin (FROMKNECHT; VELICANU, 2014) que foi desenvolvido utilizando a *Blockchain*, que fornece um serviço de Domain Name Service (DNS), NameCoin (Namecoin Team, 2022) como base. Esta solução apresenta um grau muito maior de complexidade no momento em que seu objetivo é desenvolver uma rede de *Blockchain* especifica para ser utilizada como PKI. Além disso seu desenvolvimento necessitaria de um infraestrutura de hardware muito mais robusta.

Entretanto, assim como este projeto, exitem outras soluções que fazem uso de *Smart Contracts* para desenvolver uma PKI na *Blockchain*. Este é o caso de Patsonakis *et al.* (2018) que faz apenas uma análise de como seria a implementação de uma PKI utilizando *Smart Contracts*. Patsonakis *et al.* (2020) já faz sua própria implementação utilizando a linguagem Solidity, porém focando mais em questões de como desenvolver acumuladores criptográficos eficientes. Por fim Al-Bassam (2016) apresenta uma solução muito mais próxima da apresentada neste projeto, porém sem utilizar qualquer interface gráfica ou sistema de votação para garantir maior confiabilidade.

**Tabela 1 – Comparação entre os projetos de PKI na *Blockchain***

| | Efetivamente desenvolvido e testado | Faz uso de *Smart Contracts* | Faz uso de interface gráfica | Apresenta alta complexidade |
|---|---|---|---|---|
| **PKC** | ✓ | ✓ | ✓ | X |
| **CertCoin** | ✓ | X | X | X |
| **Patsonakis *et al.* (2018)** | X | X | X | ✓ |
| **Patsonakis *et al.* (2020)** | ✓ | X | X | ✓ |
| **Al-Bassam (2016)** | ✓ | ✓ | X | X |

**Fonte: Autoria própria.**

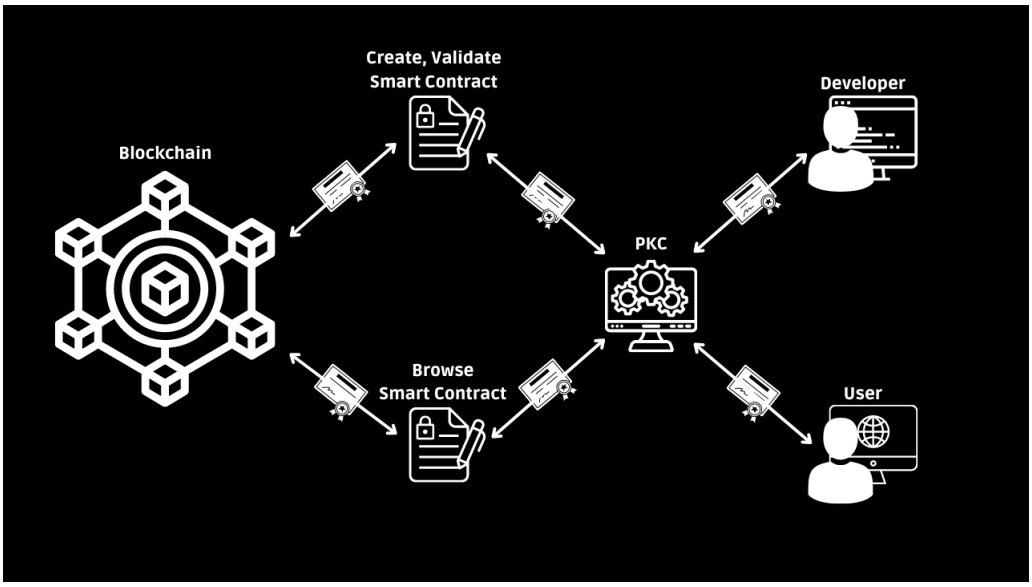
# 4 METODOLOGIA

Neste capítulo será apresentada toda a metodologia utilizada no desenvolvimento do projeto, dês de sua proposta até seus resultados finais.

## 4.1 Proposta

Este projeto se propõe a apresentar uma alternativa para os sistemas de PKI que se baseiam em CAs. Entretanto, para isso, é necessário algo com o mesmo nível, ou maior, de disponibilidade, irrefutabilidade e confiabilidade que uma CA tradicional.

**Figura 3 – Diagrama básico de funcionamento da**




**Fonte: Autoria própria.**

A fim de alcançar este nível de disponibilidade e irrefutabilidade foi utilizado o auxilio da tecnologia de *Blockchain*. Como as redes *Blockchain* fazem uso de mecanismos de consenso, operações inválidas ou de adulteração de dados se tornam computacionalmente, ou financeiramente, inviáveis.

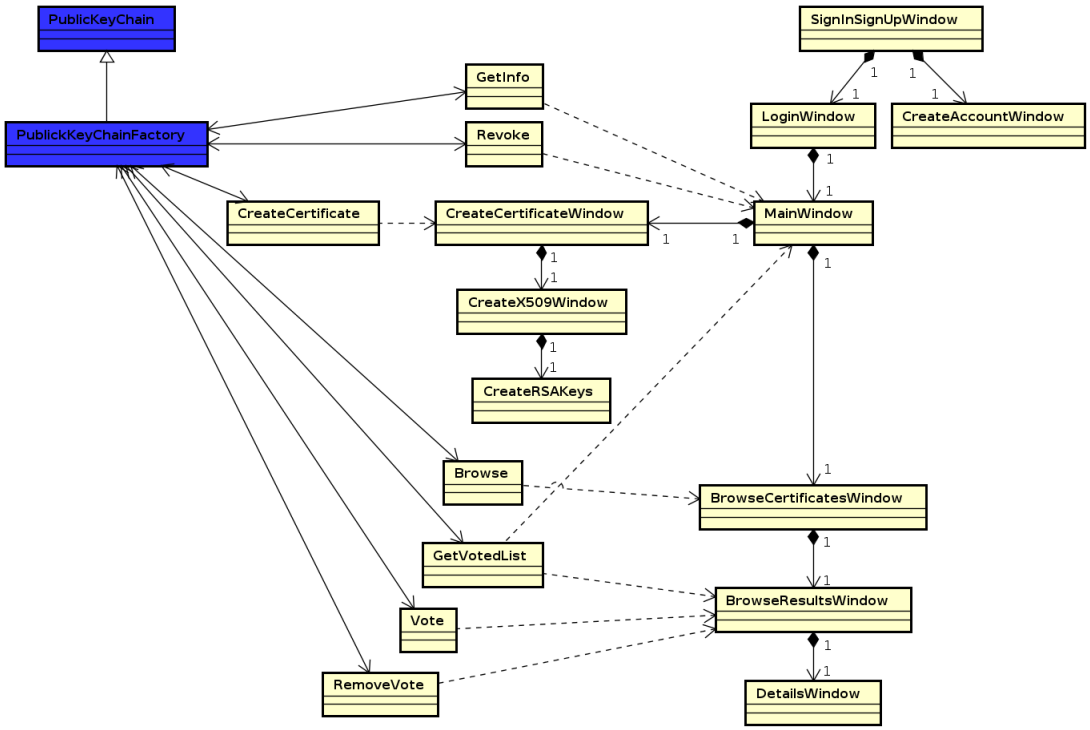
Para garantir a confiabilidade foi criado um sistema de validação de certificados baseado em votos, onde os usuários do sistema podem votar em um certificado que considerem confiável aumentando assim sua reputação. Desta forma os certificados com mais votos são considerados mais confiáveis e aparecem em primeiro nos resultados de uma busca.

Outra funcionalidade que se deseja alcançar é o desenvolvimento de um sistema gráfico de fácil uso para o usuário comum. Como a utilização de CAs se dá de forma praticamento transparente para o usuário, criar um sistema que adicione complexidade para atividades básicas, como acessar *websites*, se opõe a ideia deste ser uma alternativa a estes sistemas.

## 4.2 Implementação

A implementação do projeto se deu, em sua maioria, com a linguagem Python em vista de ser uma linguem orientada a objetos, com ampla documentação e com suporte ao desenvolvimento da lógica, além da interface gráfica do projeto.

**Figura 4 – Diagrama de Classes Simplificado do projeto**




**Fonte: Autoria própria.**

Para implementar uma PKI em uma *Blockchain* foi decidido se utilizar da tecnologia de *Smart Contracts* para criar as regras e armazenar os dados relacionados a cada certificado digital. Existem redes com inúmeras plataformas que permitem o uso de *Smart Contracts*, desta forma foi feita uma comparação entre as principais *Blockchain* que suportam o uso de *Smart Contracts*.

**Tabela 2 – Comparação entre as *Blockchains* que suportam *Smart Contracts***

| | **Ethereum** | **Fabric** | **Corda** | **Stellar** | **Rootstock** |
|---|---|---|---|---|---|
| **Ambiente de execução** | EVM | Docker | JVM | Docker | VM |
| **Turing completude** | Turing completa | Turing completa | Turing incompleta | Turing incompleta | Turing completa |
| **Utilização dos *Smart Contracts*** | geral | geral | moeda digital | moeda digital | moeda digital |
| **Linguagens suportadas** | Solidity, Serpent, LLL, Mutan, Vyper | Java, Golang | Java, Kotlin | Python, Javascript, Golang, PHP | Solidity |
| **Permissão** | publica | privada | privada | consórcio | publica |
| **Algoritmos de consenso** | PoS (PoW anteriormente) | PBFT | Raft | SCP | PoW |
| **Modelagem de dados** | baseado em contas | par chave-valor | baseado em transações | baseado em contas | baseado em contas |

**Fonte: Adaptado de Cointelegraph (2022).**

Ao analisar a tabela é possível perceber que a maior parte das redes estão limitadas a aplicações financeiras ligadas a moedas digitais, como é o caso de Corda, Stellar e Rootstock.

As únicas duas redes que permitem o desenvolvimento de aplicações gerais são Ethereum e Fabric, contudo Fabric é uma rede privada onde seu uso só pode ser feito por usuários autenticados. Desta forma a *Blockchain* Ethereum se mostrou a mais ideal e viável para o desenvolvimento do projeto.

Outro ponto que precisou ser decidido foi a linguagem que os *Smart Contracts* seriam escritos, visto que existem inúmeras linguagem desenvolvidas justamento para este proposito. No fim a linguagem escolhida foi Solidity em vista de ser mais conhecida e apresentar a documentação mais completa em comparação com outras linguagens. A discussão a respeito das vulnerabilidades da Solidity e sua escolha ao invés da linguagem mais moderna Vyper será vista no capitulo 5.2.

### 4.2.1 Funcionalidades

As principais funcionalidades elencadas para garantir o funcionamento do sistema como uma PKI são as seguintes.

#### 4.2.1.1 Criação e Armazenamento de Certificados Digitais

O sistema deve funcionar, de maneira simplificada, como um banco de certificados digitais, os usuários são capazes de armazenar seus certificados na *Blockchain*. O *Smart Contract* recebe os arquivos *.crt* dos certificados e armazena seus bytes na rede.

Esta operação é imutável, ou seja, uma vez que o certificado for armazenado na *Blockchain* não é mais possível modifica-lo. Isto evita questões de segurança e fraudes.

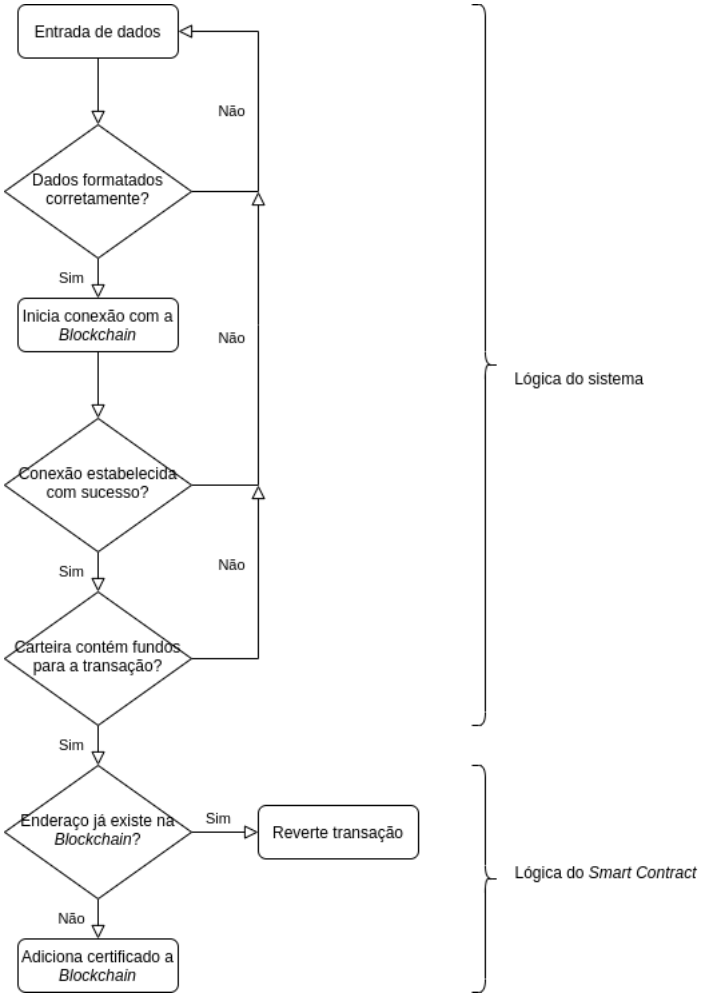
Para facilitar o uso, um sistema de criação dos arquivos *.crt* para certificados no estilo X509 também foi implementado no sistema.

#### 4.2.1.2 Validação de Certificados Digitais

Para garantir a confiabilidade do sistema é utilizado um sistema de validação, ou votação. Esta funcionalidade tem como objetivo informar os usuários quais certificados são mais confiáveis do que outros. Desta forma os usuários são capazes de validar os certificados que considerem verdadeiros ou confiáveis.

Esta funcionalidade funciona como um contador simples de votos que mantém um recorde de seus validantes, garantido assim apenas um voto por usuário. Diferentemente da operação de criação de certificados, está não é imutável e pode ser revertida, possibilitando assim que o usuário retire seus votos em casos de certificados falsos ou comprometidos.

**Figura 5 – Fluxograma da operação de criação**




**Fonte: Autoria própria.**

#### 4.2.1.3 Pesquisa e Instalação de Certificados Digitais

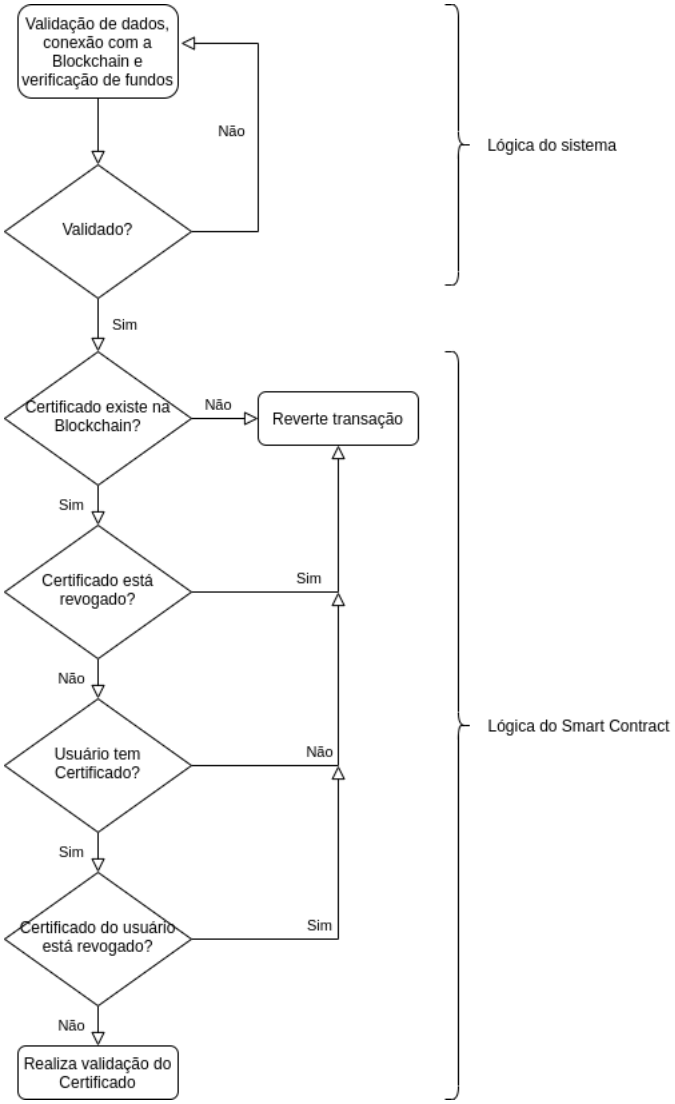
Como o sistema deve ser prático tanto para desenvolvedores quanto para usuários comuns, uma funcionalidade de busca de certificados na *Blockchain* foi implementada. Esta busca funciona de maneira similar a busca DNS, onde o usuário informa a Uniform Resource Locators (URL) e o sistema irá retornar seu certificado correspondente.

Contudo o sistema não bloqueia a criação de mais de um certificado para uma mesma URL. Nestes casos a pesquisa irá retornar uma lista contendo todos os certificados criados para esta URL ordenados por quantidade de validações (votos), de forma a facilitar o usuário a encontrar os certificados mais confiáveis.

Outra funcionalidade que ajuda na praticidade do sistema é a instalação automática dos certificados tanto no sistema operacional quanto nos navegadores. Para isso o sistema faz uso de um *script* em Bash, o que também implica nesta funcionalidade ser exclusiva para sistemas Linux.

Também é possível desinstalar os certificados que foram instalados anteriormente.

**Figura 6 – Fluxograma da operação de validação**



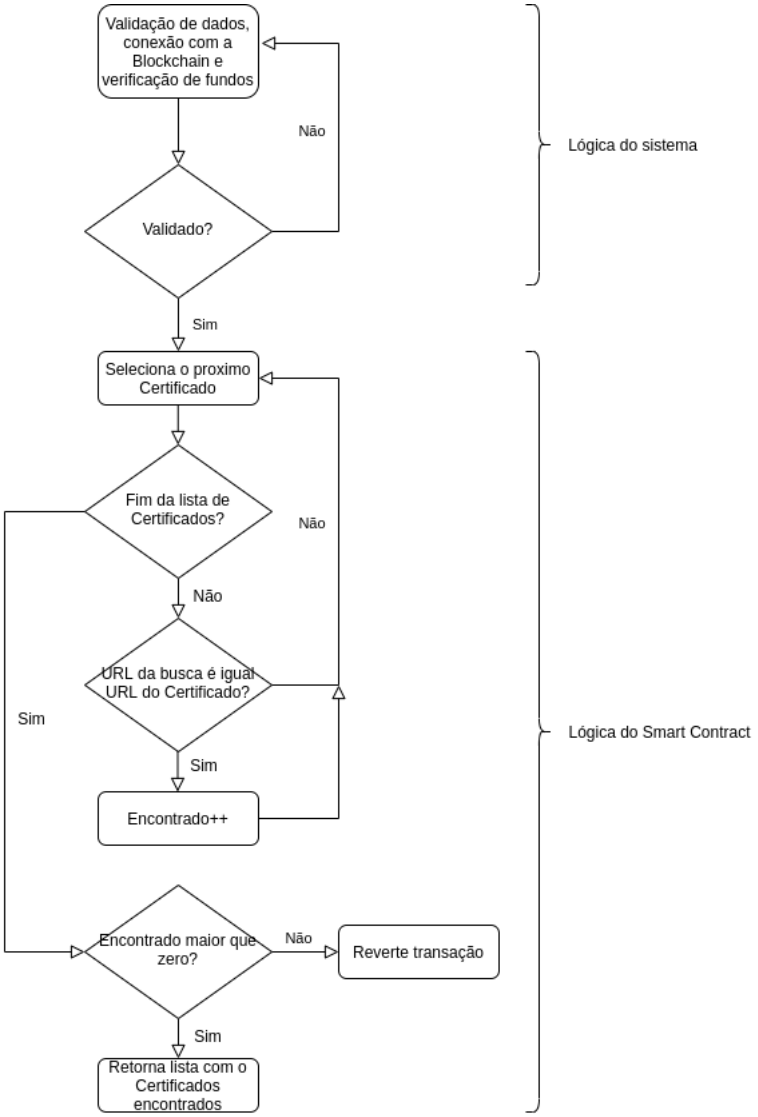

**Fonte: Autoria própria.**

#### 4.2.1.4 Lista de Confiança

A lista de confiança é uma funcionalidade opcional para o usuário que tem como objetivo facilitar e melhorar a experiencia de uso do sistema. O usuário pode adicionar os certificados que considera altamente confiáveis em sua lista de confiança, esta lista permite que o usuário faça buscas mais refinadas utilizado a opção "estou com sorte"onde apenas os certificados validados por aqueles que estão em sua lista de confiança serão exibidos.

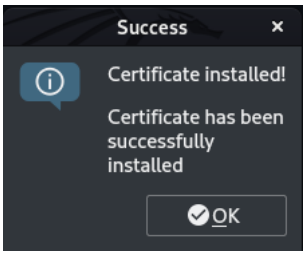
Além disso também é possível fazer uma instalação em massa de todos os certificados validados por aqueles que estão em sua lista de confiança utilizado a opção "atualizar a partir da lista de confiança".

**Figura 7 – Fluxograma da operação de pesquisa**




**Fonte: Autoria própria.**

**Figura 8 – Informe que o certificado foi instalado com sucesso**

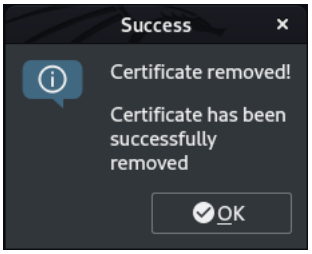



**Fonte: Autoria própria.**

### 4.2.2 Interface Gráfica

Como discutido anteriormente a praticidade e facilidade de uso, principalmente para o usuário comum, são um dos focos principais do desenvolvimento deste projeto. Desta forma o sistema foi desenvolvido com uma interface gráfica navegável. Cada uma de suas principais janelas serão discutidas a seguir.

**Figura 9 – Informe que o certificado foi desinstalado com sucesso**




**Fonte: Autoria própria.**

#### 4.2.2.1 Tela de Cadastro e Login

**Figura 10 – Tela de Cadastro e Login**


**Fonte: Autoria própria.**

A primeira janela que é aberta ao executar o sistema é a tela para cadastro ou login. Nesta tela existem apenas três botões:

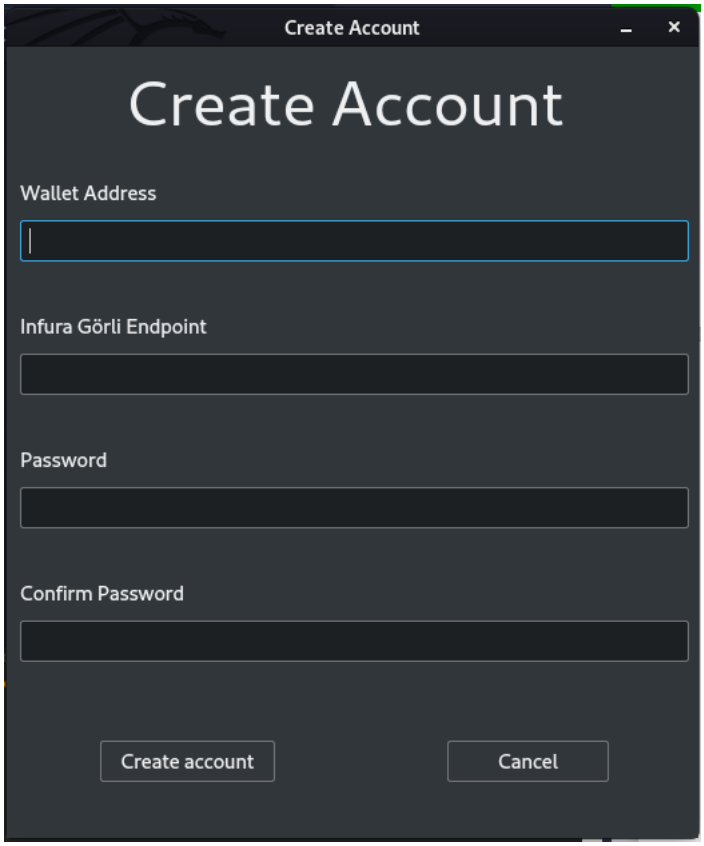
- *Login*: abre a janela para fazer o login;
- *Create account*: abre a janela para fazer o cadastro;
- *Quit*: fecha o sistema.

#### 4.2.2.2 Tela de Cadastro

A janela de cadastro permite que o usuário se cadastre no sistema. Este cadastro não faz parte da comunicação com a *Blockchain*, apenas serve como comodidade para evitar que o usuário entre todos as suas informações sempre que for utilizar o sistema. Nesta tela existem quatro entradas de texto e dois botos:

- *Wallet Address*: endereço publico da carteira Ethereum;
- *Infura Görli Endpoint*: endpoint da API *Infura* para a Blockchain *Görli* (mais informações no cap 5.2);

**Figura 11 – Tela de Cadastro**




**Fonte: Autoria própria.**

- *Password*: senha para acessar o sistema;
- *Confirm Password*: confirmação da senha por segurança;
- *Create account*: valida as informações entradas e realiza o cadastro no sistema;
- *Cancel*: volta para a tela anterior.

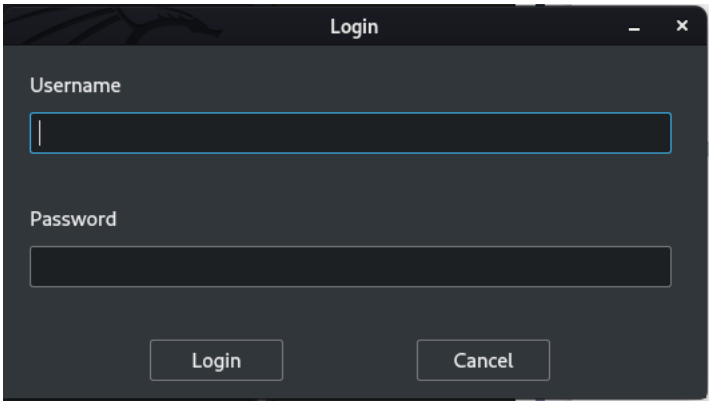
As informações de cadastro são salvas em um arquivo chamado *accounts.csv*. Neste arquivo são armazenados o endereço publica da carteira Ethereum em texto simples como chave primaria, a segunda função *hash* da senha e o *endpoint* cifrado por criptografia utilizando a primeira função *hash* da senha como chave.

#### 4.2.2.3 Tela de Login

A janela de login permite acesso as funcionalidades do sistema. Essa tela apresenta duas entradas de texto e dois botões:

- *Username*: endereço publico da carteira Ethereum;
- *Password*: senha para acessar o sistema;

**Figura 12 – Tela de Login**

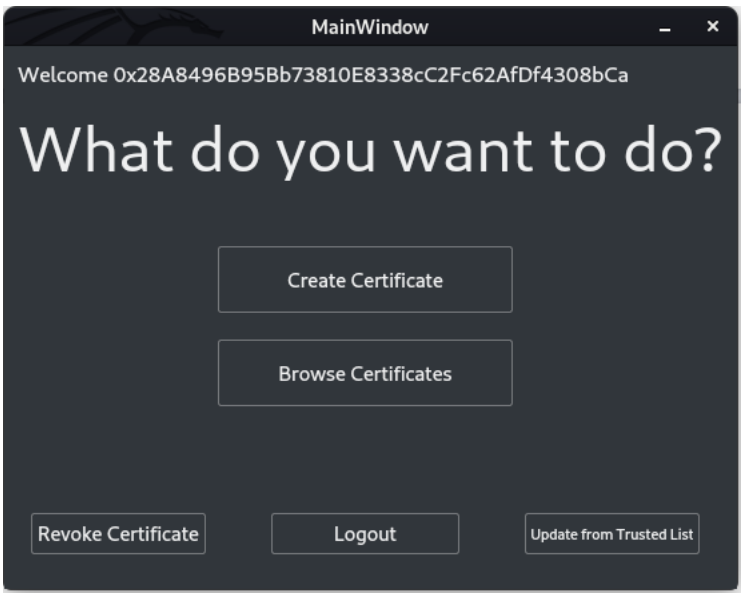



**Fonte: Autoria própria.**

- *Login*: confirma as informações e realiza o login;
- *Cancel*: volta para a tela anterior.

#### 4.2.2.4 Tela Principal

**Figura 13 – Tela Principal**




**Fonte: Autoria própria.**

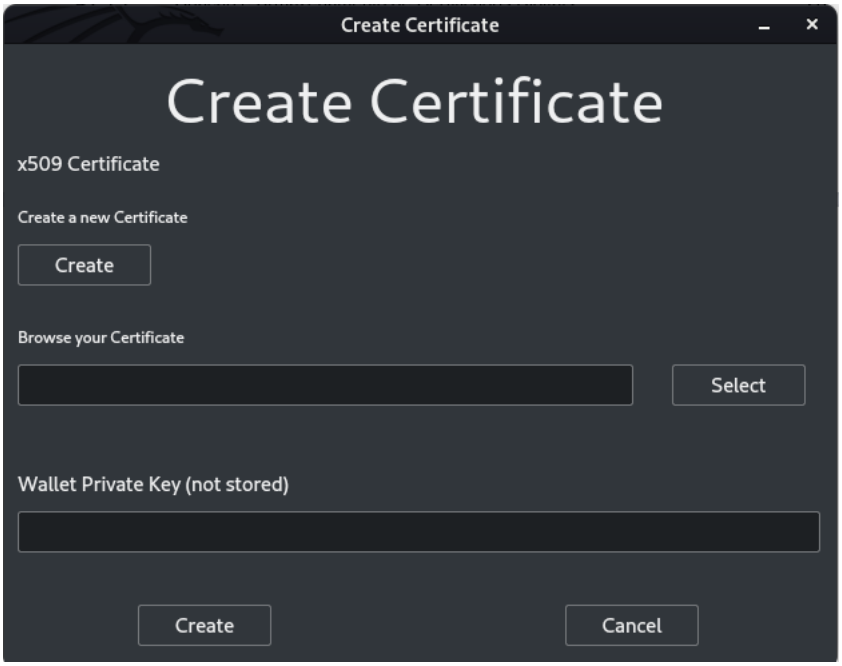
A janela principal do sistema da acesso a todas as suas funcionalidades. Nesta tela existem cinco botões:

- *Create Certificate*: abre a janela para fazer a criação de Certificados;
- *Browse Certificate*: abre a janela para fazer a busca de Certificados;
- *Revoke Certificate*: realiza a operação de *Revoke* do próprio Certificado, inutilizando-o;

- *Update from Trusted List*: realiza a instalação em massa de todos os Certificados validados por aqueles que estão em sua lista de confiança;
- *Logout*: faz o logout e volta para a tela anterior.

#### 4.2.2.5 Tela de Criação de Certificado

**Figura 14 – Tela de Criação de Certificado**



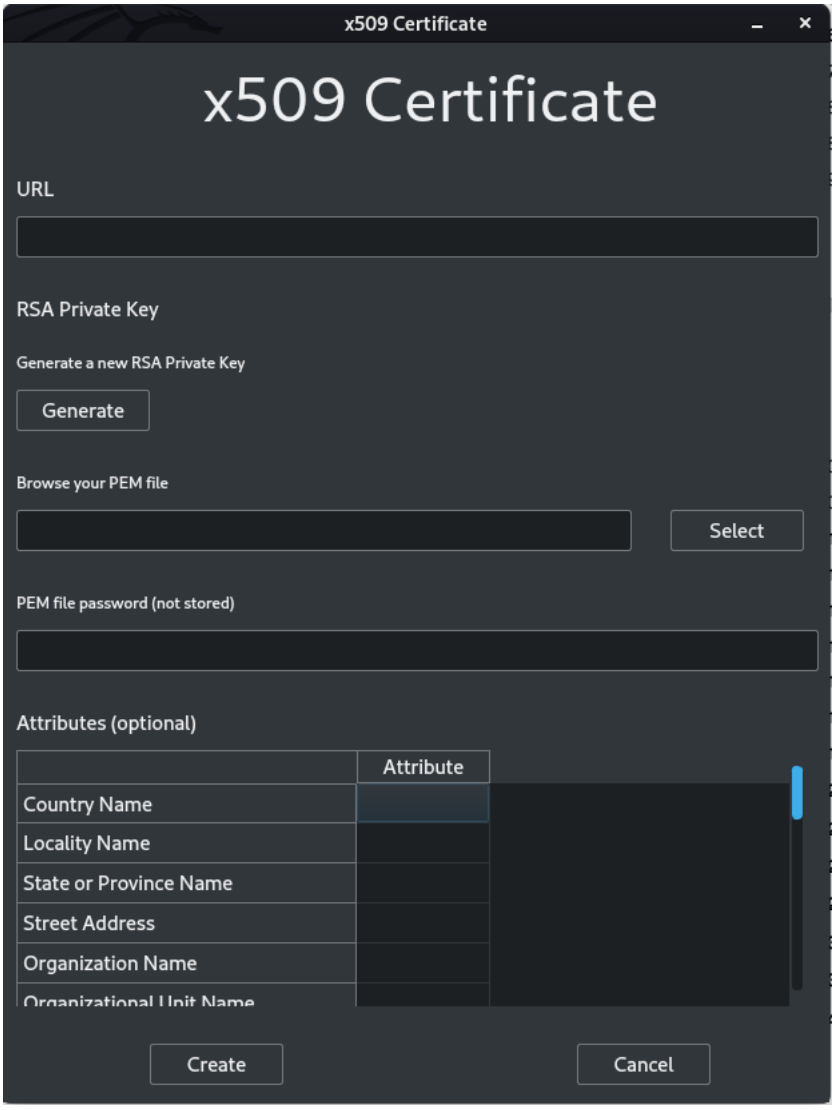

**Fonte: Autoria própria.**

A janela de criação de certificado permite que o usuário suba na *Blockchain* seu arquivo *.crt* de certificado digital X509. Nesta tela existem quatro botões e duas entradas de texto:

- *Create a new Certificate*: abre a janela para gerar um novo arquivo de Certificado X509;
- *Browse your Certificate*: caminho para o arquivo do Certificado;
- *Select*: abre o navegador de arquivos para facilitar a entrada do caminho do arquivo;
- *Wallet Private Key*: chave privada da carteira Ethereum cadastrada, necessária fara realizar pagamentos. Como esta operação irá inserir dados na Blockchain é cobrada uma pequena taxa chamada de GAS;
- *Create*: valida os campos e faz a inserção dos dados na Blockchain;
- *Cancel*: volta para a tela anterior.

A janela de geração de certificados X509 serve como uma facilitador para o usuário que não precisa utilizar ferramentas externas para gerar seu arquivo *.crt*. Nesta tela existem quatro botões, três entradas de texto e uma tabela:

**Figura 15 – Tela de Criação de Certificados X509**




**Fonte: Autoria própria.**

- *URL*: URL que identifica o Certificado, é automaticamente atributo como o atributo *Common Name* do Certificado;
- *Generate a new RSA Private Key*: abre a janela para geração de par de chaves RSA;
- *Browse your PEM file*: caminho para o arquivo da chave RSA;
- *Select*: abre o navegador de arquivos para facilitar a entrada do caminho do arquivo;
- *PEM file password*: senha para abrir o arquivo *.pem* contendo a chave RSA;
- *Attributes*: tabela opcional que permite a entrada dos principal atributos para um Certificado X509;
- *Create*: valida os campos, gera o arquivo *.crt* e o salva no diretóŕio do usuário;
- *Cancel*: volta para a tela anterior.

**Figura 16 – Tela de Criação de par de chaves RSA**

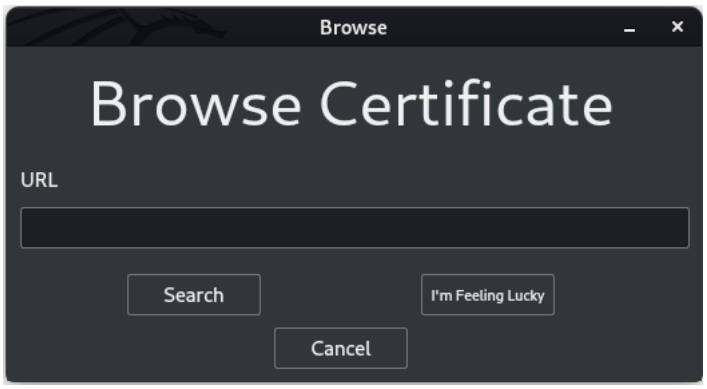

**Fonte: Autoria própria.**

A janela de geração de par de chaves RSA também é um facilitador para o usuário. Nesta tela existem dois botões e uma entrada de texto:

- *Key password*: senha para proteção do arquivo;
- *Generate*: gera um par de chaves RSA com a senha informada e o salva no diretório do usuário;
- *Cancel*: volta para a tela anterior.

#### 4.2.2.6 Tela de Pesquisa

**Figura 17 – Tela de Pesquisa**



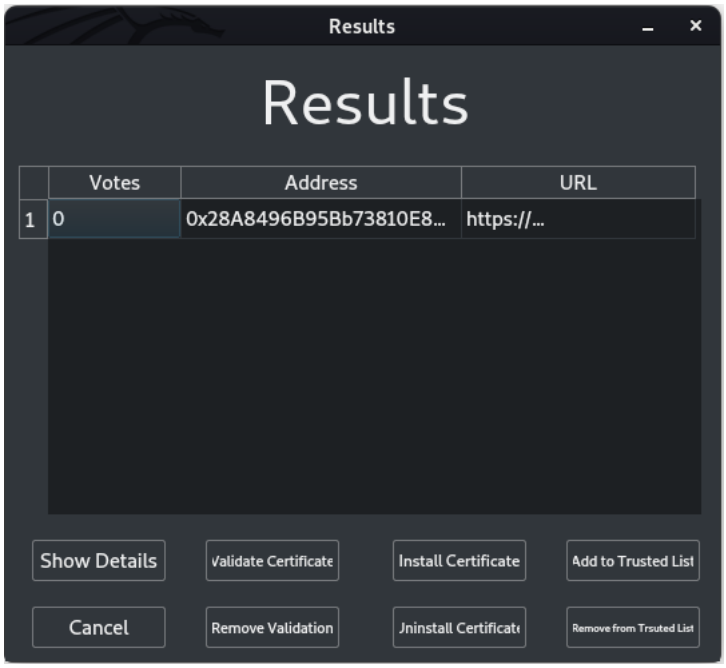

**Fonte: Autoria própria.**

A janela de busca de certificados permite que o usuário entre com uma URL e retorna uma lista de todos os certificados criados para esse endereço. Nesta tela existem três botões e uma entrada de texto:

- *URL*: URL que se deseja pesquisar;
- *Search*: realiza a pesquisa da URL informada;

- *I'm Feeling Lucky*: realiza a pesquisa da URL, porém apenas Certificados validados por aqueles que estão em sua lista de confiança serão exibidos;
- *Cancel*: volta para a tela anterior.

**Figura 18 – Tela de Resultados da Pesquisa**



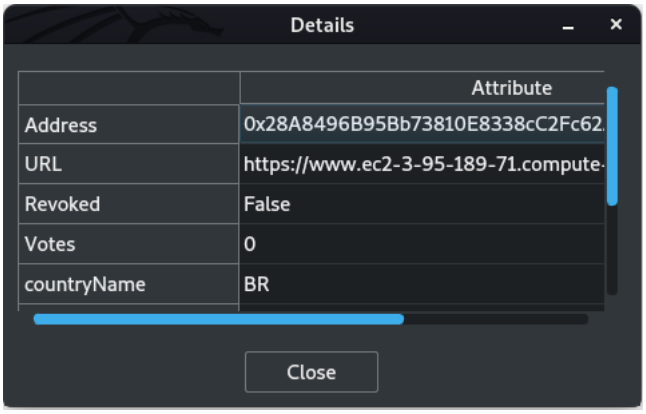

**Fonte: Autoria própria.**

A janela de resultados da pesquisa lista todos os certificados que foram encontrados nesta pesquisa. Ao selecionar qualquer linha da tabela, novos botões surgem com as funcionalidades disponíveis para cada certificado. Nesta tela existem oito botões e uma tabela:

- Tabela de resultados: contém a lista de todos os Certificados criados com a URL pesquisada. A lista é ordenada por numero de votos;
- *Show Details*: abre a janela de detalhes do Certificado;
- *Validade Certificate*: executa a funcionalidade de validação para o Certificado selecionado, um *PopUp* pedindo a chave privada da carteira será aberto, visto que é uma operação de alteração da *Blockchain*;
- *Install Certificate*: instala o Certificado na máquina do usuário e em todos os seus navegadores;
- *Add to Trusted List*: adiciona o Certificado a Lista de Confiança do usuário.
- *Remove Validation*: reverta a validação para o Certificado selecionado, um *PopUp* pedindo a chave privada da carteira será aberto, visto que é uma operação de alteração da *Blockchain*;

- *Uninstall Certificate*: desinstala o Certificado da máquina, bem como de todos os navegadores;
- *Remove from Trusted List*: Remove o Certificado da Lista de Confiança;
- *Cancel*: volta para a tela anterior.

**Figura 19 – Tela de Detalhes do Certificado**




**Fonte: Autoria própria.**

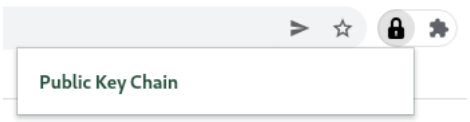
A janela de detalhes do certificado lista todas as informações a respeito do certificado selecionado. Nesta tela existe uma tabela e um botão:

- Tabela de resultados: contém o endereço do autor do Certificado, sua URL, seu status de *Revoked*, quantos votos tem e todos os seus atributos;
- *Close*: volta para a tela anterior.

### 4.2.3 Extensão para Navegador

A segunda parte que complementa o sistema de PKI é uma extensão para navegadores web baseados em Chromium.

**Figura 20 – Extensão de Navegador**

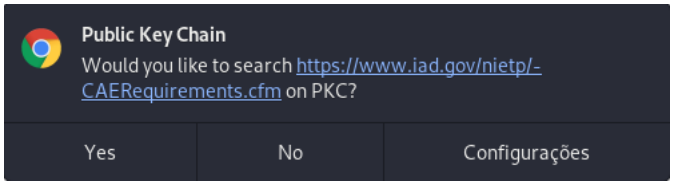



**Fonte: Autoria própria.**

A extensão não apresenta nenhuma interface gráfica ou interação com o usuário, seu funcionamento se dá automaticamente ao detectar o erro *NET::ERR_CERT_AUTHORITY_INVALID*, que é retornado pelo navegadores quando o certificado solicitado por algum site não está instalado. Ao detectar este erro um *PopUp* se abre para o usuário perguntando se gostaria

de pesquisar este site em especifico no Public Key Chain (PKC). Caso a opção "sim"seja selecionada uma tela de login do sistema é apresentada, por questões de segurança, e uma vez feito o login a janela de busca é aberta automaticamente com os resultados.

**Figura 21 – *PopUp* da Extensão**




**Fonte: Autoria própria.**

# 5 RESULTADOS E DISCUSSÕES

Neste capítulo sera apresentado todo o caminho trilhado para alcançar os resultados resultados obtidos com desenvolvimento do projeto.

## 5.1 Testes

Inicialmente os testes dos *Smart Contracts* foram feitos utilizando o simulador de *Blockchain* Ganache (ConsenSys Software Inc, 2022). Estes testes foram mais para garantir que os arquivos não apresentavam nenhum problema ou erro de compilação, visto que o funcionamento do sistema por completo só poderia ser feito em uma rede real. A utilização de uma rede principal para fazer o desenvolvimento se faz totalmente inviável, visto que a *Blockchain* Ethereum está sendo utilizada, apenas 1 criptoativo está custando mais de *R$8.000,00*.

Durante a maior parte do desenvolvimento do projeto a rede de testes utilizada foi a rede de testes Rinkeby devido a sua maior quantidade de *faucets* ativas e utilização dos mecanismos de consenso PoW e PoS. Entretanto, no dia 25 de Setembro de 2022, a *Blockchain* Ethereum, base da rede de testes Rinkeby, realizou a transação de seu mecanismo de consenso PoW para PoS. Após essa transação foi decidido que apenas duas redes de testes seriam mantidas e as outras seriam descontinuadas até 2023. As redes que seriam mantidas são Görli e Sepolia, enquanto isso a rede Rinkeby já tem data para ser descontinuada. Desta forma, já no fim do desenvolvimento, foi decidido utilizar a rede de testes Görli para evitar quaisquer problemas que possam existir com as redes descontinuadas.

## 5.2 Dificuldades

As primeiras dificuldades se fizeram na utilização da linguagem Solidity que era desconhecida pelo autor. Além disso a criação da interface gráfica também se mostrou desafiadora visto que foi feita em sua integridade na linguagem Python.

Outra dificuldade encontrada foi a limitação de hardware para utilização de um nó completo da *Blockchain*. Para sincronizar um nó completo da rede de testes Rinkeby, por exemplo, os requisitos mínimos de hardware pedem um Solid State Drive (SSD) de pelo menos 1Tera Byte (TB) de armazenamento com uma velocidade de escrita de pelo menos 68Mega Byte (MB) por segundo e 39,1MB por segundo de velocidade de leitura. Desta forma foi decidido que, devido ao projeito ser um protótipo, a utilização do serviço de nós hospedados Infura.

Por fim a utilização da linguagem Solidity como base do projeto também causou certas dificuldades. A ideia seria de utilizar a linguagem mais moderna e segura Vyper para realizar todo o desenvolvimento dos *Smart Contracts*. Porém devido a linguagem ser muito recente não há uma documentação tão completa quando Solidity, além de haver bem menos discussões

e fóruns a seu respeito. A linguagem Vyper, no momento atual, também não apresenta todas as funcionalidades da linguagem Solidity, por exemplo a linguagem não suporta a criação de vetores de strings, e isto também foi um fator que levou a sua não adesão.

## 5.3 Resultados Obtidos

O teste final do sistema foi feito simulando sua utilização por dois usuário diferentes, um desenvolvedor web e um usuário comum. Para isso, um certificado foi gerado no sistema e instalado em um servidor web simples na nuvem. Em seguida um usuário cadastrado tentava-se acessar a URL deste servidor web em outra máquina (que também havia o sistema instalado), o que naturalmente gerou a mensagem de erro *NET::ERR_CERT_AUTHORITY_INVALID*. Em seguida a extensão para navegador era ativada automaticamente e perguntava para o usuário se este gostaria de pesquisar a URL no sistema. Após clicar em “sim” e realizar o login, a pesquisa era feita automaticamente e retornava a lista contendo o certificado necessário. O usuário faz a instalação deste certificado e reinicia seu navegador. Por fim ao acessar a mesma URL agora o site se mostra normalmente com navegação segura.

# 6 CONCLUSÃO

O projeto, inicialmente, tinha como ideia “propor um novo modelo de certificado” e armazenar apenas as chaves públicas e atributos na *Blockchain*. Entretanto isso também implicaria na necessidade de um novo protocolo Hyper Text Transfer Protocol (HTTP) seguro apenas para este novo modelo de certificado. Com o andar do desenvolvimento do projeto foi visto que essa ideia era inviável, e que apenas armazenar certificados do modelo X509 autoassinados na *Blockchain* e instala-los automaticamente na máquina e navegadores do usuário.

A partir deste momento o desenvolvimento do projeto ficou muito mais claro e factível. Os primeiros testes realizados com o auxilio do servidor web hospedado na nuvem demonstraram que o sistema precisava dar mais atenção para a formatação do atributo *Common Name* que estava sendo simplesmente populado com a URL informada pelo usuário. Neste caso por mais que o certificado fosse instalado corretamente em todos os navegadores, um novo erro era gerado (NET::ERR_CERT_COMMON_NAME_INVALID), este erro se dá quando o atributo *Common Name* do certificado não bate com a URL que se está tentado acessar. Para estes casos era necessário fazer o parse apenas das informações de domínio (e sub domínio caso exista) da URL informada pelo usuário.

Uma vez que estes problemas foram sendo resolvidos, a utilização do sistema foi se mostrando cada vez melhor. Por fim o sistema completo com auxilio da extensão para navegador foi utilizado corretamento com sucesso e se mostrou bem viável para utilização como uma alternativa para CAs, principalmente para novos desenvolvedores que desejam criar aplicações confiáveis e seguras mas não tem condições de contratar CAs confiáveis.

## 6.1 Comentários Adicionais

O projeto desenvolvido está disponível em sua totalidade no GitHub, podendo ser acessado através do endereço https://github.com/Matias157/PKC.

Além disso um vídeo de demonstração de seu funcionamento também está disponível no YouTube através do endereço https://youtu.be/R--6mKlFBQ8.

## 6.2 Trabalhos Futuros

Como já discutido no capitulo 5.2, o desenvolvimento dos *Smart Contracts* se fez utilizando a linguagem Solidity devido a sua maior e mais completa documentação. Entretanto, a linguagem Solidity apresenta algumas vulnerabilidades conhecidas que foram corrigidas no desenvolvimento da linguagem Vyper. Um possível trabalho futuro seria refazer os *Smart Contracts* deste projeto em linguagem Vyper, garantindo assim um sistema mais confiável e seguro.

# REFERÊNCIAS


ADAMS, C.; LLOYD, S. **Understanding PKI: Concepts, Standards, and Deployment Considerations**. 2nd. ed. USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0672323915.

AL-BASSAM, M. **Trustery: A Public Key Infrastructure and Identity System Based on Smart Contracts on the Blockchain**. 2016.

American Institute of Certified Public Accountants (AICPA) and Canadian Institute of Chartered Accountants. **WebTrust principles and criteria for certification authorities, Version 1.0, February 9, 2000; Exposure draft (American Institute of Certified Public Accountants), 2000, February 9**. USA: AICPA Professional Standards, 2000. 496 p.

ANGELIS, S. D. *et al.* **PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain**. 2017.

ARNBAK, A.; EIJK, N. van. Certificate authority collapse: Regulating systemic vulnerabilities in the https value chain. **2012 TRPC Conference**, 2012.

BERKOWSKY, J.; HAYAJNEH, T. **Security issues with certificate authorities**. 2017. 449-455 p.

BOEYEN, S. *et al.* **Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile**. [*S.l.*], 2008. (Request for Comments, 5280). Disponível em: https://www.rfc-editor.org/info/rfc5280.

BUTERIN, V. **A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM**. 2015.

BUTERIN, V.; GRIFFITH, V. **Casper the Friendly Finality Gadget**. 2017.

COINTELEGRAPH. **A deep dive into the 5 popular smart contract development platforms and their comparison**. 2022. Disponível em: https://cointelegraph.com/blockchain-for-beginners/smart-contract-development-platforms. Acesso em: 21 set. 2022.

ConsenSys Software Inc. **Ganache**. 2022. Disponível em: https://trufflesuite.com/ganache/. Acesso em: 21 set. 2022.

CRIS. **Diagram of a public key infrastructure**. 2022. Disponível em: https://en.wikipedia.org/wiki/Public_key_infrastructure#/media/File:Public-Key-Infrastructure.svg. Acesso em: 21 set. 2022.

FROMKNECHT, C.; VELICANU, D. **CertCoin : A NameCoin Based Decentralized Authentication System 6 . 857 Class Project**. 2014.

GARREAU, M. **A Developer's Guide to Ethereum, Pt. 1**. 2022. Disponível em: https://snakecharmers.ethereum.org/a-developers-guide-to-ethereum-pt-1/. Acesso em: 21 set. 2022.

HOUSLEY, R. *et al.* **Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile**. [*S.l.*], 2002. (Request for Comments, 3280). Disponível em: https://www.rfc-editor.org/info/rfc3280.

Infura Inc. **Infura**. 2022. Disponível em: https://infura.io/. Acesso em: 21 set. 2022.

MARQUEZ, G. **Quanto custa um certificado digital: veja valores, tipos e onde comprar**. 2018. Disponível em: https://nfe.io/blog/assinatura/quanto-custa-certificado-digital/. Acesso em: 21 set. 2022.

NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. **Cryptography Mailing list at https://metzdowd.com**, 03 2009.

Namecoin Team. **Namecoin**. 2022. Disponível em: https://www.namecoin.org/. Acesso em: 21 set. 2022.

NOFER, M. *et al.* Blockchain. **Business Information Systems Engineering**, v. 59, 03 2017.

PATSONAKIS, C. *et al.* Implementing a smart contract pki. **IEEE Transactions on Engineering Management**, PP, p. 1–19, 04 2020.

PATSONAKIS, C. *et al.* **Towards a Smart Contract-Based, Decentralized, Public-Key Infrastructure: 16th International Conference, CANS 2017, Hong Kong, China, November 30—December 2, 2017, Revised Selected Papers**. [*S.l.*: *s.n.*], 2018. 299-321 p. ISBN 978-3-030-02640-0.

RESCORLA, E. **The Transport Layer Security (TLS) Protocol Version 1.3**. [*S.l.*], 2018. (Request for Comments, 8446). Disponível em: https://www.rfc-editor.org/info/rfc8446.

Solidity Team. **Solidity**. 2022. Disponível em: https://soliditylang.org/. Acesso em: 21 set. 2022.

The go-ethereum Authors. **Go Ethereum**. 2022. Disponível em: https://geth.ethereum.org/. Acesso em: 21 set. 2022.

Vyper Team. **Vyper**. 2022. Disponível em: https://vyperlang.org/. Acesso em: 21 set. 2022.