UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CAIO CASTILHO DONATO REGAL

SEGURANÇA EM WEB3 : VULNERABILIDADES EM CARTEIRAS DIGITAIS BASEADAS EM NAVEGADORES

BAURU
Janeiro/2023

CAIO CASTILHO DONATO REGAL

SEGURANÇA EM WEB3 : VULNERABILIDADES EM CARTEIRAS DIGITAIS BASEADAS EM NAVEGADORES

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Me. Luiz Felipe de Camargo

BAURU
Janeiro/2023

R333s

Regal, Caio Castilho Donato

Segurança em Web 3 : Vulnerabilidades em carteiras digitais baseadas em navegadores / Caio Castilho Donato Regal. -- Bauru, 2023

35 p.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Faculdade de Ciências, Bauru

Orientador: Luiz Felipe de Camargo

1. Proteção de dados. 2. Criptografia de chaves públicas. 3. Criptografia de dados (Computação). 4. Computadores Medidas de segurança. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências, Bauru. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Caio Castilho Donato Regal

Segurança em WEB3 : Vulnerabilidades em carteiras digitais baseadas em navegadores

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Me. Luiz Felipe de Camargo
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Profa. Dra. Simone Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Prof. Dr. Kelton A. Pontara da Costa
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Bauru, 20 de janeiro de 2023.

Dedico esse texto a quem busca evoluir e melhorar seu meio, aos curiosos que criam e transformam.

Agradecimentos

Agradeço aos meus pais, Antonio Carlos e Elizeth pelo amor, carinho, apoio e paciência em toda a minha vida.

Agradeço a professora Marcia Aparecida Zanoli pela ajuda e orientação durante o curso na Unesp Bauru.

Agradeço ao professor e meu orientador Luiz Felipe de Camargo por todo o tempo dedicado a me ajudar e orientar.

Nós somos aquilo que fazemos repetidamente. Excelência, então, não é um modo de agir, mas um hábito.

Will Durant

Resumo

A aplicação de carteira de criptomoedas mais utilizada da rede Ethereum, Metamask, possui uma brecha que pode ser manipulada para ter acesso não permitido a uma conta. É apresentada essa falha, assim como é feito uma apresentação sobre os processos envolvidos com a chave privada e a chave publica utilizadas pela Blockchain, como elas são geradas e armazenadas pela aplicação de navegadores Metamask, além de expor métodos de criptografia como a criptografia de curva elíptica e a criptografia assimétrica. São propostas mudanças na aplicação para reparar esse problema de segurança, como o uso do endereço MAC do computador para ser usado na criptografia, apesar de conseguir fazer uma prova do conceito o resultado não é o suficiente para ser implementado na aplicação real

Palavras-chave: Ethereum, Metamask, Navegadores, Sandbox, Segurança da informação, Criptografia, Criptografia de curva elíptica.

Abstract

Metamask is the most used crypto wallet in the Ethereum network, it has a flaw that enables unauthorized access to an existing account. This security flaw is shown and discussed in this study where it is also introduced the process behind Private and public key generation as well as how it is stored in the browser metamask application in addition to exploring how cryptographic such as Elliptic curve cryptography and Asymmetric cryptography are developed. Furthermore it is presented suggestions to fix this security flaw like using the computer's MAC adress as part of the encryption. This was developed as proof of concept, but results are not good enough to be implemented in the real application

Keywords: Ethereum, Metamask, Browsers, Sandbox, Information Security, Cryptography, Elliptic Curve Cryptography.

Lista de figuras

Lista de abreviaturas e siglas

RSA	Rivest - Shamir - Adleman
mod	Módulo
SHA	Secure Hash Algorithm
SHA256	Secure Hash Algorithm 256 bits
BIP39	Bitcoin Improvement Proposal 39
ECSDA	Elliptic Curve Digital Signature Algorithm
IV	Initialization Vector
PBKDF	Password-based Key Derivation Function
AES128	Advanced encryption standard 128 bits
AES GCM	Advanced encryption standard galois/counter mode
DBIR	Data Breach Investigations Report
MAC	Media Access Control
API	Application Programming Interface
IMARC	The International Market Analysis Research and Consulting Group

Sumário

1 Introdução

A tecnologia conhecida como *blockchain* foi primeiramente apresentada por Satoshi Nakamoto, no seu artigo *"Bitcoin: A Peer-to-Peer Electronic Cash System"*, onde foi estruturada a parte matemática para o *bitcoin*. A tecnologia de *blockchain* não é só a fundação de todas as criptomoedas como também encontrou aplicação no mercado de financeiro tradicional (PIERRO, 2017).

Segundo o relatório da IMARC (The International Market Analysis Research and Consulting Group), o mercado de criptomoedas alcançou 1,78 trilhões de dólares em 2021 e tem sua previsão para que alcance 32 trilhões em 2027 (GLOBENEWSWIRE, 2022). Com um mercado desse tamanho e boas expectativas é importante que novos trabalhos sejam feitos sobre o tema para a sua melhor compreensão.

A corretora Foxbit define *blockchain* como um livro público que registra as transações de criptomoedas, de caráter confiável e imutável. Em outras palavras, a *blockchain* é responsável por registrar as informações de valores ou quantidades de moedas transacionadas, data, local de transação e dados dos usuários (FOXBIT, 2022).

A principal diferença desse sistema para o sistema financeiro convencional é a descentralização. Em Nakamoto e Bitcoin (2008) é relatado que o comércio na internet depende quase que exclusivamente em instituições financeiras que servem como um sistema de confiança terceirizado, uma entidade que facilita a troca entre duas partes que confiam nesse terceiro, tornando-o dessa forma em um sistema centralizado.

A característica de um sistema financeiro descentralizado é que não existe uma central de autoridade. Isso significa que seus usuários precisam tomar cuidados especiais e buscar corrigir possíveis erros de segurança para tornar o todo deste sistema melhor.

1.1 Justificativa

A segurança da informação vem afetando cada vez mais as rotinas de usuário de tecnologia, Souza (2021) levanta dados sobre isso: "O Brasil sofreu nada menos do que 8,4 bilhões de tentativas de ataques cibernéticos ao longo de 2020, sendo que, deste montante, 5 bilhões ocorreram apenas nos últimos três meses do ano (outubro, novembro e dezembro). É isso que aponta o mais novo relatório do Fortiguard Labs".

É evidente que existem inúmeras vulnerabilidades em computadores, possibilitando assim, diversos ataques. Dessa forma, programas, aplicativos e sistemas passaram a ser termos comuns na linguagem de grande parte das pessoas, entretanto, termos negativos, como vírus, invasões, *malwares*, etc., vieram em conjunto. Regan (2022) define: "O termo *malware* se

refere a software que danifica dispositivos, rouba dados e causa o caos. Existem muitos tipos de *malware*, vírus, cavalos de Tróia, *spyware*, *ransomware*, etc.".

De acordo com o relatório da Team (2022), o roubo de criptomoedas em 2021 foi de aproximadamente 3.2 bilhões de dólares. Um aumento de 516% quando comparado com 2020. A área de Segurança apresenta pouco desenvolvimento acadêmico quando comparada com outras áreas da computação. Até mesmo no curso de graduação em Ciências da Computação pouco é discutido sobre segurança dentro de aplicações ou da informação.

Dentro da área de segurança é de extrema importância que se façam estudos de novas vulnerabilidades e que as mesmas tenham visibilidade para que possam ser corrigidas e evitadas no futuro. Dessa forma o estudo que será feito nesse projeto é uma forma de mostrar novas vulnerabilidades, criar novas discussões acadêmicas e gerar conhecimento sobre o assunto.

Já existe uma solução para o problema apresentado nesse projeto, mas envolve o uso de carteiras de *hardware*, que são pagas e podem ter alto custo para o usuário comum, assim como podem ser ineficazes para alguns usos automatizados. A solução proposta é algo desenvolvido no *software* e será gratuita para o usuário.

Para completar, o estudo de Khande et al. (2021) faz uma análise de problemas relacionados ao uso de senhas e analisa um relatório da Verizon Data Breach Investigations Report (DBIR) de 2019 que contém dados que apontam que 80% das invasões com violação de dados ocorre por senhas ruins ou comprometidas e que pelo menos 29% de todos os *hacks* independente do tipo de ataque acontece por conta de credenciais roubadas.

1.2 Objetivo

Esse trabalho tem como objetivo expor e propor uma correção para um erro de segurança no armazenamento das chaves de autorização no aplicativo de carteira de criptomoedas Metamask que está sendo abusado em crimes cibernéticos.

1.2.1 Objetivo Geral

Aumentar a segurança de aplicações como Metamask, criando uma camada adicional de segurança ao método de criptografia e de verificação de segurança.

1.2.2 Objetivos Específicos

- Estudar os processos de criptografia envolvidos na *Blockchain*;
- Estudar os processos de criptografia envolvidos na criação e armazenamento das chaves da rede Ethereum;

- Estudar o funcionamento do Metamask;
- Selecionar a informação para ser usada como identificação única do computador onde a Metamask se encontra instalada;
- Desenvolver uma prova de conceito que possa impedir o uso do *Vault* em um computador que não o gerou;

1.3 Organização da Monografia

Este trabalho está estruturado em cinco capítulos. O capítulo 1 apresenta a introdução, detalhando a justificativa, os objetivos e a estrutura do trabalho. No capítulo 2 é apresentada a fundamentação teórica que serve como base do trabalho. O capítulo 3 apresenta o problema a ser abordado. O capítulo 4 expõe o processo de desenvolvimento da solução e seus resultados e o capítulo 5 apresenta as conclusões obtidas do trabalho.

2 Fundamentação Teórica

2.1 *Blockchain*

A ideia de um sistema financeiro descentralizado digital já existia antes do *bitcoin* e da *blockchain* com alguns protocolos *e-cash* nos anos de 1980 e 1990, que utilizavam técnicas de criptografia Chaumian Blinding, mas que falharam por que dependiam de um intermediário centralizado. Mais tarde foi criado o *b-money* que também falhou, pois não conseguiu provar em como faria um consenso descentralizado (BUTERIN, 2014).

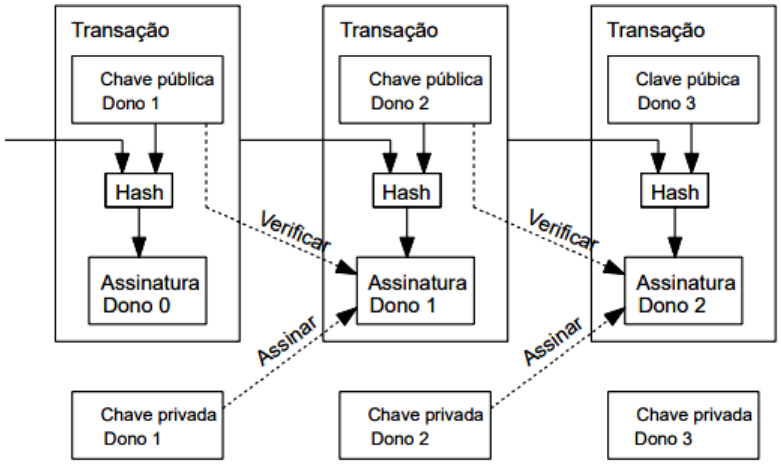
Em Nakamoto e Bitcoin (2008) é dito que existe uma necessidade de um sistema eletrônico de pagamento que seja baseado em prova criptográfica em vez de um sistema de confiança terceirizado, que permita que duas partes possam fazer transações diretamente, sem que exista necessidade de uma terceira parte. O *bitcoin* foi o primeiro sistema financeiro descentralizado que obteve sucesso. Combinando formas já estabelecidas primitivas para gerenciar a posse por meio da criptografia de chave pública com um algoritmo de consenso para registrar quem é dono de cada moeda, conhecido como "prova de trabalho".

A *blockchain* é como um livro razão digital que armazena qualquer tipo de dados (RODECK; SCHMIDT, 2022). Diferente dos bancos de dados tradicionais, a *blockchain* é descentralizada, ou seja ao invés de ser mantido em apenas um local por um administrador centralizado, várias cópias idênticas do banco de dados da *blockchain* estão armazenadas em inúmeros computadores na rede. Cada um desses computadores é identificado como nó. Para Rodeck e Schmidt (2022) o nome *blockchain* não é acidental. A *blockchain* é descrito como uma "corrente" formada por "blocos" de dados. Conforme novos dados são adicionados à rede, um novo "bloco" é criado e anexado à "corrente". Isso requer que todos os nós atualizem suas versões da *blockchain* para que sejam iguais.

Define-se uma moeda eletrônica como uma corrente de assinaturas digitais. Cada dono de moeda transfere a sua moeda para o outro assinando digitalmente a *hash* da transação passada e a chave pública do próximo dono, adicionando-os ao final da moeda. Pode-se verificar as assinaturas para verificar a posse das moedas na corrente (NAKAMOTO; BITCOIN, 2008).

Para entender melhor a Figura 1 nas próximas seções será explicado como funciona a criptografia de chave pública, outros detalhes de elementos do funcionamento da *blockchain* como o funcionamento das transações, prova de trabalho e consenso, tipos de nós e árvores de Merkle não são relevantes para o objeto de estudo deste trabalho.

Figura 1 – Transações *bitcoin*.



Fonte: Nakamoto e Bitcoin (2008)

2.2 Criptografia de Chave Pública

O termo surgiu de um artigo científico publicado em 1976 por Whitfield Diffie e Martin E Hellman. Nele os autores comentam que com o desenvolvimento de *hardware* digital de baixo custo, os mesmos começariam a ser utilizados para aplicações comerciais e que a tecnologia de criptografia na época não era eficiente o suficiente (DIFFIE; HELLMAN, 1976).

Para as partes interessadas em trocar informações em um canal inseguro, era necessário que houvesse uma forma de embaralhar essa mensagem e para isso era usada uma chave. O problema desse método é que ele exige que as duas partes tenham a chave e para isso é necessária uma forma segura de compartilhar essa chave, na época isso seria uma forma física de compartilhamento. Desse modo aumentava o custo e gerava atrasos na comunicação entre essas partes, o que acabava criando uma barreira para as partes que procuravam privacidade, eficiência e segurança na troca de informações (DIFFIE; HELLMAN, 1976).

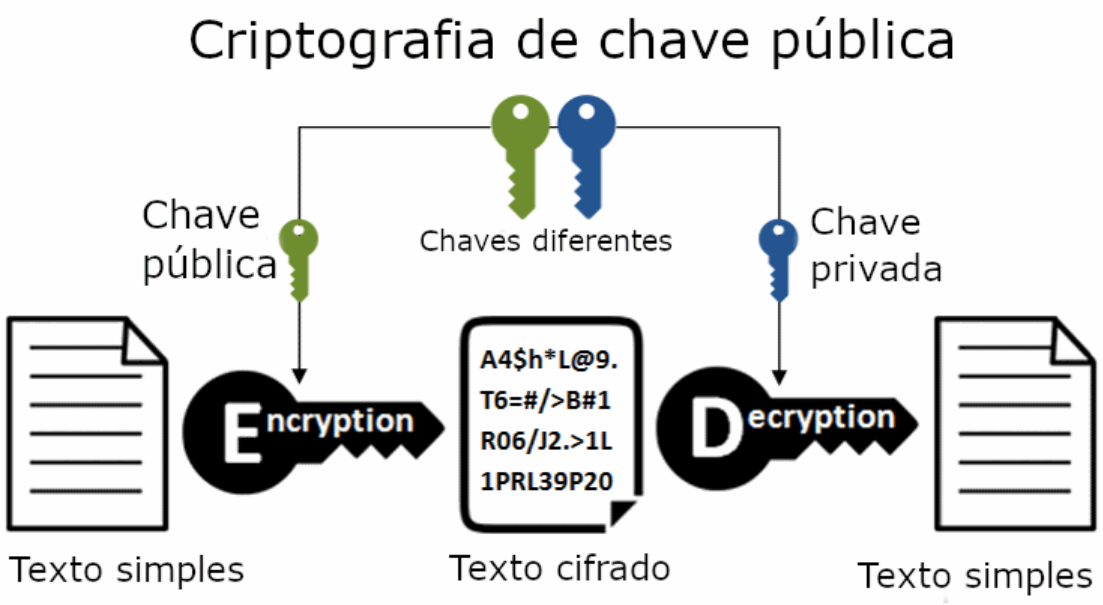
A criptografia de chave pública ou criptografia assimétrica foi criada para resolver esse problema. Ela utiliza um par de chaves que são baseadas em funções matemáticas conhecidas como funções arapucas e tem uma propriedade especial: são fáceis de computar em uma direção, mas difíceis de computar na direção oposta (ANTONOPOULOS; WOOD, 2018).

Para exemplificar essas funções será utilizado o número 63.808.741. Dado apenas o número, encontre dois fatores primos que multiplicados tenham ele como resultado. Encontrar estes dois valores é algo complexo, porém inversamente, multiplicar 7219 por 8839 é algo trivial e leva a 63.808.741, o valor inicial. Esse tipo de problema se chama fatoração prima e possui a propriedade especial citada anteriormente, a multiplicação é simples porém encontrar os fatores é complexo.

As chaves são divididas em chave pública e chave privada, a primeira é usada para

encriptar a mensagem e a última para decriptar a mensagem, como demonstrado pelo diagrama simplificado na Figura 2 (NAKOV, 2018). Dessa forma é possível compartilhar apenas a chave pública e outra parte poderá usá-la para escrever uma mensagem que somente quem possuir a chave privada poderá ler.

Figura 2 – Criptografia assimétrica.



Fonte: Nakov (2018)

A criptografia de chave pública é usada em diferentes criptossistemas como o RSA (Rivest - Shamir - Adleman), Diffie-Hellman, criptografia de curva elíptica e seus derivados, entre outros. As principais *blockchains* (Bitcoin e Ethereum) utilizam-se de criptografia de curva elíptica e como dito na Introdução, esse estudo foca em uma vulnerabilidade da Metamask, uma carteira da rede Ethereum, dessa forma a próxima seção propõe um detalhamento maior sobre criptografia de curva elíptica.

2.2.1 Criptografia de Curva Elíptica

A criptografia de curva elíptica é baseada no problema de logaritmo discreto, ela permite o funcionamento das capacidades dos criptossistemas assimétricos via encriptação, assinaturas e troca de chaves. É considerada a sucessora do sistema RSA pois é mais ágil na hora de criar chaves, assinar, além de possuir chaves menores (NAKOV, 2018).

Na matemática, curvas elípticas são planos de curvas algébricas, na criptografia pode-se usar a equação de Weierstrass dada por

$$y^2 = x^3 + Ax + B \tag{2.1}$$

onde A e B são constantes. É necessário especificar a quem A, B,x,y pertencem (WASHINGTON, 2008). Neste estudo utiliza-se A,B,x,y $\in \mathbb{F}$p, onde $\mathbb{F}\mathbf{p}$ é um campo finito composto pelo conjunto de inteiros módulo $\mathbf{p}$, onde $\mathbf{p}$ é um número primo, dessa forma utiliza-se a aritmética modular e não a regular, assim sendo todas as operações (como soma e multiplicação de pontos) dentro do campo produzirão um ponto resultante dentro do campo.

A função de curva elíptica utilizada pela Ethereum e o *bitcoin* se chama secp256k1, por utilizarem a mesma curva elíptica é possível reutilizar ferramentas e bibliotecas em ambas as redes. A curva foi estabelecida pelo Instituto Nacional de Padrões e Tecnologia dos Estados Unidos e é definida pela seguinte função :

$$y^2 = (x^3 + 7) \tag{2.2}$$

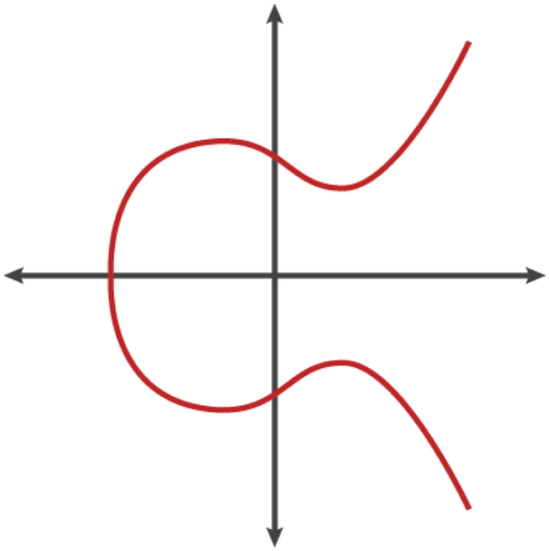
ou

$$y^2 mod \mathbf{p} = (x^3 + 7) mod \mathbf{p} \tag{2.3}$$

O mod $\mathbf{p}$ indica que essa curva $\in \mathbb{F}\mathbf{p}$, onde $\mathbf{p} = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ (ANTONOPOULOS; WOOD, 2018).

A curva elíptica pode ser observada na Figura 3.

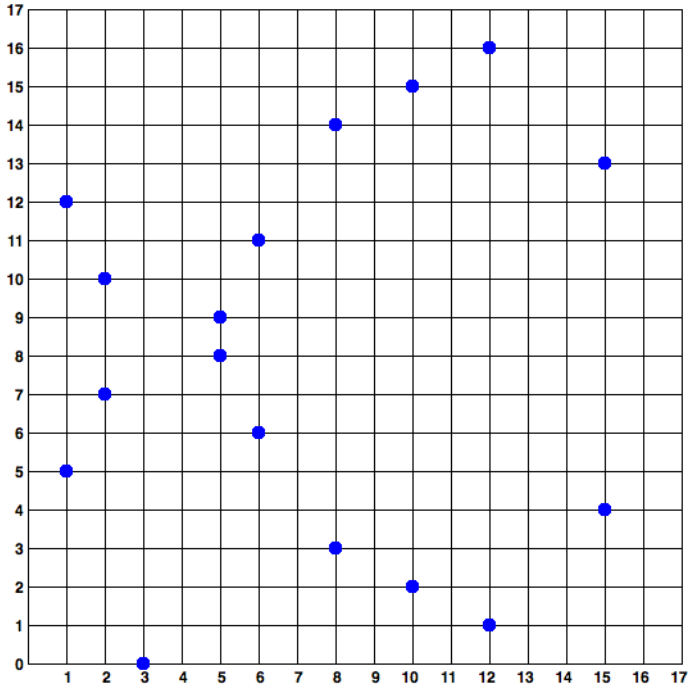
Figura 3 – Visualização de uma curva elíptica.



Fonte: Antonopoulos e Wood (2018)

A Figura 4 mostra um "curva simplificada"com $\mathbf{p} = 17$ representada pelos pontos azuis. Na prática a curva é na verdade um conjunto de pontos na matriz quadrada $\mathbf{p}x\mathbf{p}$, não uma curva clássica. Essa curva produz chaves muito curtas de 4-5 bits, quando normalmente as curvas produzem chaves de 256 bits ou mais.

Figura 4 – Visualização de uma curva elíptica com $\mathbf{p} = 17$.



Fonte: Antonopoulos e Wood (2018)

Fica fácil então calcular se um certo ponto pertence a curva. Para isso o ponto $\mathbf{Q}(x, y) \in$ a curva $y^2 = (x^3 + 7)$ (mod $\mathbf{17}$) se e somente se :

$$x^3 + 7 - y^2 = 0(mod17) \tag{2.4}$$

Portanto o ponto $\mathbf{Q}(6, 11) \in y^2 = (x^3 + 7)$ (mod $\mathbf{17}$) pois $(6^3 + 7 - 11^2\ )\%\ 17 = 0$, já o ponto $\mathbf{J}(9, 15) \notin y^2 = (x^3 + 7)$ (mod $\mathbf{17}$) pois $(9^3 + 7 - 15^2\ )\%\ 17\ \neq 0$ (NAKOV, 2018).

Após entender se um ponto pertence ou não a curva, é atrativo também entender que ao somar dois pontos da curva o resultante será outro ponto na curva, e assim deve ser considerado o operador de adição em curvas elípticas. Dessa forma, dados os pontos $\mathbf{P_1}$ e $\mathbf{P_2}$ pertencentes a curva, existe $\mathbf{P_3} = \mathbf{P_1} + \mathbf{P_2}$ que também será pertencente a curva (ANTONOPOULOS; WOOD, 2018).

Com a ideia da adição em mente, se adicionar um ponto da curva $\mathbf{G}$ a ele mesmo o resultado será $\mathbf{G} + \mathbf{G} = 2$ * $\mathbf{G}$, e ao adicionar novamente G ao resultado se obtém 3 * $\mathbf{G}$ (NAKOV, 2018). Assim pode-se definir a multiplicação de um ponto na curva como *k* * $\mathbf{G} = \mathbf{G} * \mathbf{G} * \mathbf{G} * \mathbf{G} * \mathbf{G}$ .... *k* vezes, isso gera um ponto novo na mesma curva além de ser um processo rápido (ANTONOPOULOS; WOOD, 2018).

2.2.2 Gerando as chaves

Antes de mostrar como as chaves são geradas na rede Ethereum com uso da criptografia de curva elíptica, elas serão brevemente definidas. Nas próximas seções serão aprofundadas

suas características e funcionalidades. A chave privada pode ser considerada como um número inteiro escolhido de forma aleatória e a chave pública como ponto na curva elíptica.

O padrão secp256k1 especifica um ponto $\mathbf{G}$ como ponto gerador, que é usado em todas as implementações do padrão secp256k1, pertencente a curva e que de quem todas as chaves são geradas. Sendo assim utiliza-se a chave privada como o número *k* e o multiplica-se pelo ponto $\mathbf{G}$ para gerar um novo ponto $\mathbf{K}$, ao qual foi brevemente denominado de chave pública (ANTONOPOULOS; WOOD, 2018). Desse modo tem-se :

$$K = k * G \tag{2.5}$$

Seguindo toda a lógica até então apresentada para criptografia, produzir $\mathbf{K} = k*\mathbf{G}$ deve ser um cálculo de mão única, de *k* para $\mathbf{K}$, assim $\mathbf{K}$ ou, chave pública, pode ser compartilhada e *k*, ou chave privada, deve continuar sendo secreta (ANTONOPOULOS; WOOD, 2018).

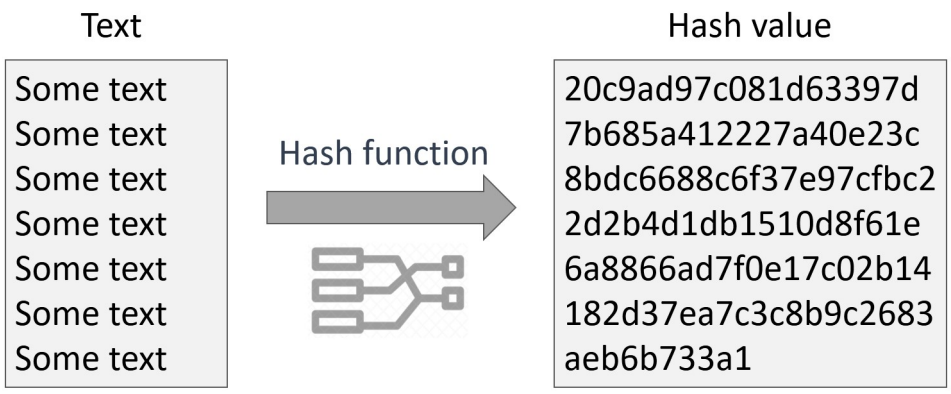
2.2.2.1 Função Hash

A função *hash* será abordada de forma sucinta nessa seção pois ela é parte da transformação da chave pública para endereços na rede Ethereum, assim como na transformação do número aleatório em chave privada.

Em Dworkin (2015) é dito que funções *hash* possuem propriedades especiais, destacando a resistência à colisão e resistência à segunda pré-imagem que são muito importantes para aplicações em segurança da informação. Em funções *hash*, a entrada é chamada de mensagem e a saída é chamada de valor *hash*, a entrada pode ter tamanho variado enquanto a saída sempre terá um valor fixo. Essas funções possuem muitas aplicações em criptografia como geração de chaves, códigos de autenticação e algoritmos de números pseudo-aleatórios onde criam uma maior segurança e eficiência.

No processo de criação das chaves serão utilizadas duas funções *hash* da família SHA, a SHA256 e a keccak256, de característica importante sendo elas não reversíveis, ou seja uma vez os dados sendo processados é muito difícil conseguir a entrada. Esse processo pode ser observado na Figura 5 (ZHU, 2022).

Figura 5 – Visualização de uma função hash



Fonte: Nakov (2018)

2.3 Rede Ethereum

A rede Ethereum é uma infraestrutura descentralizada de código aberto capaz de executar programas chamados de *smart contracts* ou contratos inteligentes. Ela utiliza a *blockchain* para armazenar e sincronizar seus estados e suas mudanças, e floresce em aplicações econômicas descentralizadas fornecendo alta auditabilidade, disponibilidade, transparência e neutralidade o que também reduz ou elimina a censura (ANTONOPOULOS; WOOD, 2018).

A rede Ethereum interage com dois tipos de endereços, contas de propriedade externa (*externally owned account*) e os contratos inteligentes, que são como programas que rodam uma máquina virtual da rede. As contas de propriedade externa são as contas que uma pessoa comum usará para interagir com a rede fazendo transações (BUTERIN, 2013).

Contas de propriedade externa possuem um par de chaves, uma chave privada que controla a conta, quem a possuir terá acesso e controle total àquela conta e uma chave pública que nesse contexto é chamada de endereço da conta (ANTONOPOULOS; WOOD, 2018).

Existem inúmeros processos para gerar essas chaves e quem implementa esses processos e cuida das interações entre o usuário e a rede é uma aplicação chamada de carteira de criptomoedas. Desde a criação das chaves e seu armazenamento à assinaturas digitais a carteira facilita a utilização para o usuário final (JOKIć et al., 2019).

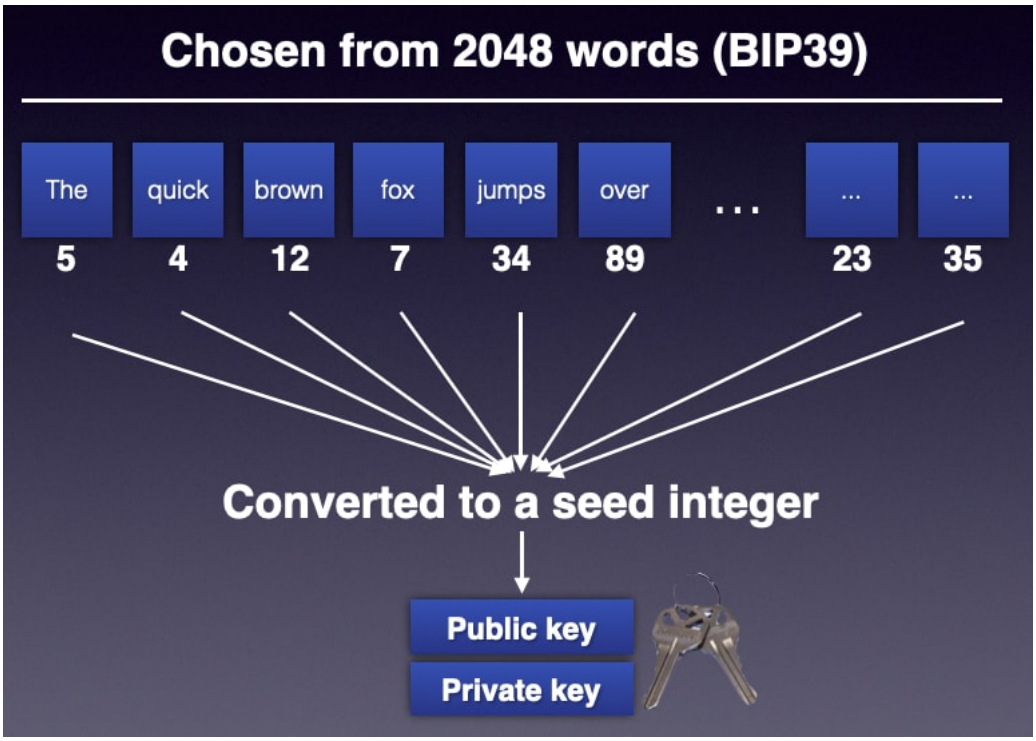
Esse estudo é focado em uma falha no armazenamento da chave privada e propõe uma solução a essa falha. Dessa forma, o foco será na carteira de criptomoedas Metamask, pois é a carteira mais utilizada.

2.3.1 Gerando Contas na Rede Ethereum

Inicialmente ao tentar criar uma conta na rede Ethereum acontece o primeiro encontro com a *seed phrase* ou frase semente. A frase semente é um conjunto de 12 mnemônicos geradas

pelo padrão BIP 39 e existe um total de 2048 mnemônicos. Cada palavra corresponde a um número e a sua ordem é importante. É de extrema importância manter essa frase segura e fazer um *backup* pois é dela que é gerada a chave privada, sem ela deveria ser impossível acessar a conta e é exatamente onde entra a falha a ser detalhada nas próximos seções. O processo de escolha das palavras e geração da frase pode ser observado na Figura 6 (LUCENA, 2022).

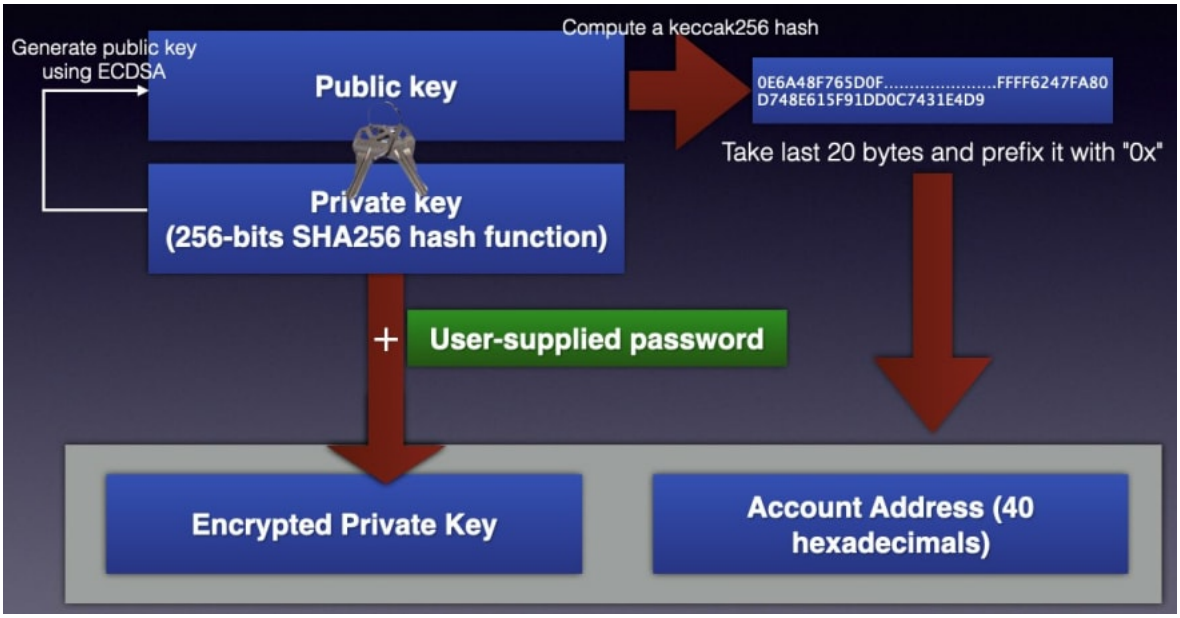
Figura 6 – Visualização BIP 39



Fonte: Lee (2021)

Em seguida os 12 mnemônicos são convertidos para um número inteiro e utilizados dentro da função *hash* SHA256 como entrada para formar a chave privada. A chave privada por sua vez será utilizada como k na função de curva elíptica para multiplicar o ponto $\mathbf{G}$ gerando o ponto $\mathbf{K}$ na curva, também conhecido como chave pública, em sequência a chave pública, por ser um ponto $(x\ y)$, é concatenada e utilizada como entrada na função *hash* keccak256, onde os últimos 20 dígitos são unidos ao prefixo 0x e finalmente é gerado o endereço da conta (LEE, 2021). Esse processo pode ser observado na Figura 7.

Figura 7 – Visualização Geração de chaves Ethereum



Fonte: Lee (2021)

3 Problema

A segurança de uma conta na rede Ethereum é totalmente dependente da frase semente e portanto da chave privada. Em um sistema descentralizado, a única forma de acessar determinada conta é com a sua frase semente, não existe um tipo de vínculo com o usuário.

Foi visto no capítulo anterior a segurança dos métodos utilizados para gerar as chaves. Em Lenstra, Kleinjung e Thomé (2013) é calculado o quanto de energia computacional seria necessário para quebrar a criptografia de curva elíptica, para dimensionar isso foi comparado com quanto de água essa mesma energia evaporaria, uma criptografia de chave de 228 bits consumiria a mesma quantidade de energia necessária para evaporar toda a água da terra. Na rede Ethereum é utilizada uma chave de 256 bits, o que implica que quebrar a criptografia de curva elíptica como não sendo algo palpável.

Considerando esse contexto, a segurança da geração das chaves e seus processos não está em dúvida, a próxima questão seria seu uso na rede e armazenamento. Em Antonopoulos e Wood (2018) é descrito que não existe encriptação nos protocolos da rede Ethereum, e que toda a comunicação entre a plataforma e os nós é não encriptada e que isso é uma necessidade para que todos possam ver tudo que está acontecendo na rede e para funcionamento do algoritmo de consenso. Dessa forma a chave privada não pode ser usada diretamente pelo sistema de forma alguma. Todo o acesso e controle dos fundos é feito por assinaturas digitais usando algoritmos com curvas elípticas. À vista disso resta explorar o armazenamento dessas chaves.

A falha de segurança que esse estudo busca propor uma solução é encontrada em carteiras de criptomoedas baseadas em navegadores, como a Metamask. Parte da segurança vem do princípio que para uma pessoa conseguir acesso a uma conta, deve se digitar a frase semente, logo se cria todo um esforço para que essa seja guardada em segurança. Se ela for digitada em um computador limpo e mantida segura, a conta deveria se manter inacessível para terceiros.

O que acontece na verdade é que a chave privada é criptografada com uma senha que é pedida para o usuário, por motivos de segurança esse arquivo é salvo no computador que é mais seguro do que centralizar todos esses arquivos em um sistema *cloud* que poderia ser atacado e possivelmente milhões de contas serem roubadas ao mesmo tempo. Por consequência, com o arquivo no computador do usuário, é possível roubar esse arquivo e a senha com uso de *malware* e ou com acesso físico ao computador, pulando totalmente a etapa que necessita da frase semente e dessa forma burlando grande parte da segurança que os usuários confiam.

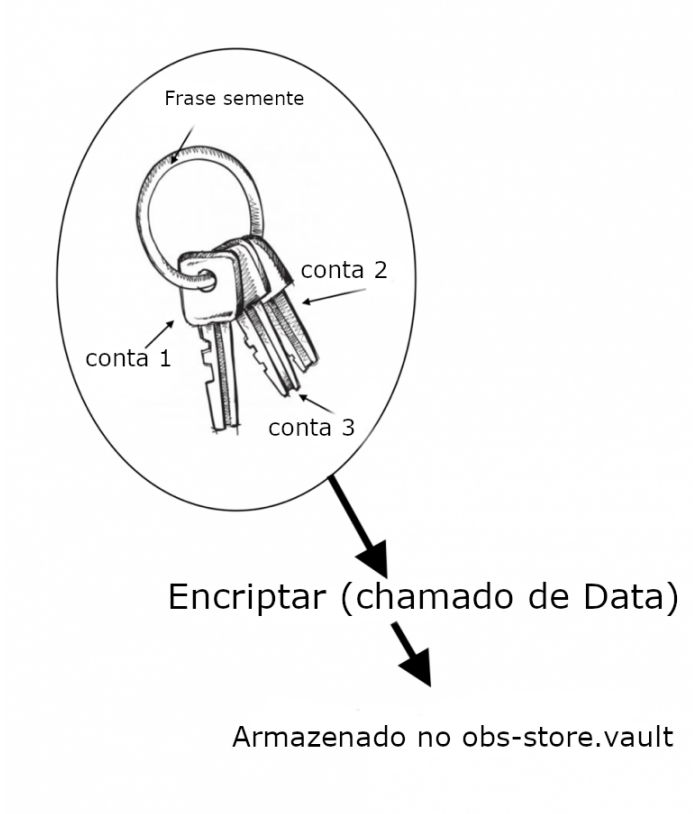
3.1 Armazenando as chaves

Para entender como funciona o problema é necessário que seja primeiro entendido como a MetaMask armazena as chaves. Em Wang (2020) o autor, que é engenheiro de segurança da empresa Certik, uma gigante da área de segurança em *blockchain*, faz uma análise sobre o código da aplicação. A maior parte do código está em código aberto e seus módulos também.

O núcleo do gerenciamento e armazenamento das contas é feito por um módulo chamado *Keyring Controller* , nele é possível criar outras contas filhas usando a chave privada como geradora. Segundo MetaMask (2022b) suas três responsabilidades são : inicializar e usar (assinando com) grupos de contas da rede Ethereum "*Keyrings*", acompanhar os nomes locais para cada uma dessas contas individuais, fornecer uma encriptação persistente com senha e restaurar as informações secretas.

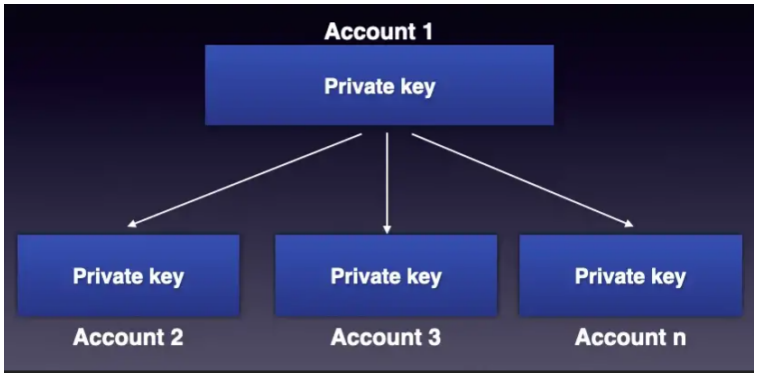
Para compreender de forma visual, pode-se imaginar *Keyring* como um chaveiro, a argola que prende as chaves é a frase semente, cada chave é uma conta individual derivada da chave privada "mãe", esse esquema pode ser observado nas Figuras 8 e 9 (LEE, 2021). Todo esse chaveiro é então encriptado com uma senha e armazenado em um local no computador chamado de *Vault* (WANG, 2020).

Figura 8 – Visualização de *Keyring*



Fonte: Wang (2020)

Figura 9 – Visualização de *Keyring* 2



Fonte: Lee (2021)

Entendido o conceito por trás do *Keyring*, pode-se acessar seu código em (METAMASK, 2022b) e observar seu funcionamento. No seu construtor observa-se o uso da classe *ObservableStore*, para criar dois objetos nomeados de *this.store* e *this.memStore*, a finalidade dessa classe é de armazenar na memória de forma síncrona um estado (DANWAHLIN, 2022).

Pode-ser observar o construtor na Figura 10 :

Figura 10 – Construtor *Keyring*

```
constructor (opts) {
  super()
  const initState = opts.initState || {}
  this.keyringTypes = opts.keyringTypes ? keyringTypes.concat(opts.keyringTypes) : keyringTypes
  this.store = new ObservableStore(initState)  --> 1
  this.memStore = new ObservableStore({  --> 2
    isUnlocked: false,
    keyringTypes: this.keyringTypes.map((krt : T ) => krt.type),
    keyrings: [],
  })

  this.encryptor = opts.encryptor || encryptor
  this.keyrings = []
}
```

Fonte: Wang (2020)

Wang (2020) afirma que *this.store* armazena o segredo da carteira de forma criptografada na memória persistente do computador, na pasta de extensões do navegador, o *Vault*. Enquanto *this.memStore* armazena de forma descriptografada o segredo da carteira na memória RAM.

No artigo de MetaMask (2023) é descrito que o *Vault* possui 3 propriedades, *Data*, que é o conteúdo criptografado com a senha, IV e *Salt*, que são parâmetros da criptografia usadas no algoritmo. O responsável por isso nos navegadores é o módulo *Browser Passworder*, ele encripta e desencripta o objeto *this.store*, utilizando o algoritmo PBKDF2 (*Password-based Key Derivation Function 2*) com o *salt* gerado pela função *crypto.getRandomValues()* para

gerar uma chave e o algoritmo AES-GCM (*Advanced encryption standard galois/counter mode)* com o IV, gerado pela mesma função *crypto.getRandomValues()*, com 10.000 iterações para realizar o processo de criptografia da *Data* (METAMASK, 2022a).

Apesar de estudos como o estudo de Kodwani, Arora e Atrey (2021) que apresenta vulnerabilidades do uso de AES-128 com PBKDFs e do estudo de Visconti et al. (2015) demonstrar vulnerabilidades do PBKDF2 em determinadas configurações, a ideia deste trabalho é de tentar voltar a premissa de que uma pessoa só deveria poder acessar a conta se ela tiver acesso a frase semente, para isso é necessário que cada *vault* apenas possa ser usado no computador de origem, pois independente da segurança e do método de criptografia utilizado no *vault* a mesma sempre será vulnerável a senhas fracas e ou repetidas, ataques utilizando *keyloggers* e ataques de surfe de ombro.

4 Desenvolvimento

A objetivo desse trabalho passa a ser então criar uma camada adicional de segurança à Metamask para garantir que o *vault* seja apenas acessado no computador em que ele foi gerado. Sendo assim é necessário primeiro identificar o computador gerador. Se o computador for identificado como não sendo o gerador do *vault*, o mesmo irá se deletar juntamente com as configurações da aplicação, retornando-a ao estado inicial de instalação.

4.1 Escolha da Chave

A chave deve ser única a cada computador e persistente, isto é, algo que não será e que não poderá ser mudado. Além disso, ela não deve pertencer aos arquivos do navegador pois estaria disposta a mesma vulnerabilidade explicada antes.

4.1.1 Endereço MAC

A escolha para a utilização foi a do endereço MAC, apesar de possíveis problemas como por exemplo um *notebook* com duas placas de rede apontaria um MAC diferente se estiver conectado por *Wi-fi* ou pelo cabo de rede *ethernet*. Outra possível questão dessa solução é que ao trocar a placa mãe ou ter algum problema com a placa de rede será necessário entrar com a frase semente novamente, o que pode ser uma possível brecha para um novo tipo de ataque.

O endereço MAC parece ser a chave mais consistente e óbvia para a solução, uma vez que se tratam de redes de *blockchain* e há necessidade de se conectar a internet para utilizar à rede, foi utilizado o comando *getmac* no *prompt* do Windows para conseguir o endereço MAC.

4.2 Sandbox do navegador

O primeiro obstáculo para obter o endereço MAC é o *sandbox* do navegador. Em Barth et al. (2008) é explicado o que navegadores baseados em Chromium, o código aberto em que o Google Chrome, Brave, Microsoft Edge entre outros se baseiam, utiliza uma arquitetura modular. Um módulo chamado *browser kernel* que interage com o computador e com o usuário, e um módulo de motor de renderização que interage com a *web*. O *sandbox* serve para que o módulo que interage com a web tenha menos privilégios limitando seus recursos e consequentemente suas ações, gerando uma segurança extra para o usuário.

A segurança do *sandbox* tão importante para os navegadores que no programa de *bug bounty* da Google pode-se ver que o maior valor pago é para vulnerabilidades envolvendo *sandbox* (GOOGLE, 2023).

4.2.1 Native Messaging

Native messaging ou mais especificamente utilizado nesse estudo, o *Chrome native messaging* é uma API do Google Chrome com a função de trocar mensagens entre aplicações e extensões do navegador. Para isso é necessário que as aplicações implementem um *native messaging host* que será responsável pelo fluxo de entrada e saída entre o computador e a extensão no processo de comunicação, o *host* será iniciado em um processo isolado (GOOGLE, 2014). Com o uso da será possível utilizar comandos necessários para conseguir o endereço MAC.

4.3 Resolução

Para evitar aumentar a complexidade da aplicação, adicionar novos módulos e outras implementações são utilizados os módulos já presentes na Metamask e antes estudados aqui, dessa maneira não há necessidade de novas auditorias de segurança em relação às ferramentas utilizadas.

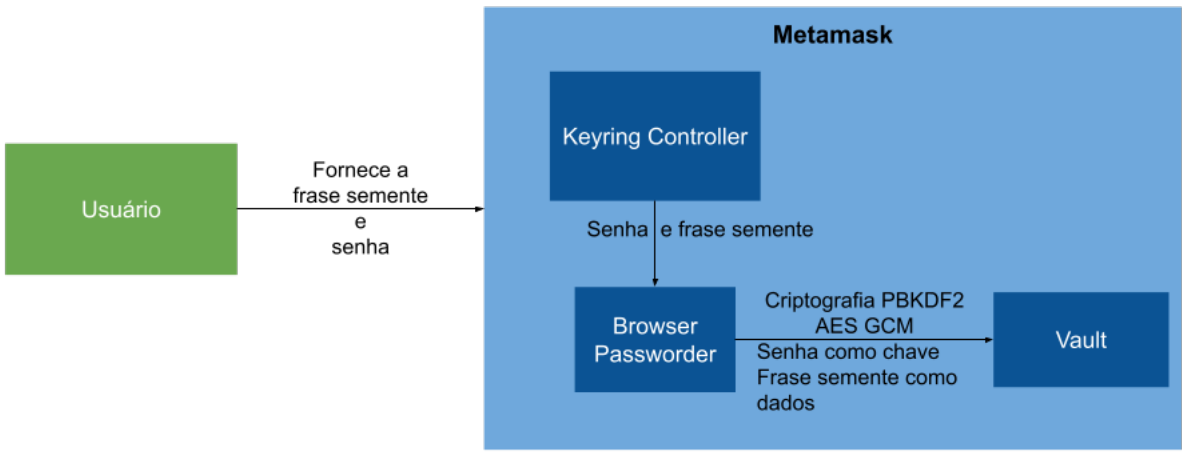
Por meio do *Chrome native messaging* é possível utilizar o comando *getmac* para conseguir o endereço MAC do computador gerador e escrever ele em arquivo, desse arquivo pode-se ler o endereço MAC e utilizar-lo como senha juntamente com o módulo *Browser Passworder*, o responsável por fazer a criptografia do *vault*. Após seu uso o arquivo deve ser apagado.

A princípio a solução aqui adotada será apenas para provar que é possível bloquear o acesso ao *vault* por outro computador que não o gerador, abrindo assim portas para estudos futuros aprimorarem a ideia, sugestões para trabalhos futuros serão discutidas na conclusão.

Com o endereço MAC a disposição da Metamask, é feita uma concatenação entre ele e a senha do usuário para gerar a criptografia por meio do *Browser Passworder*. Essa implementação não é recomendada e não deve ser realizada em aplicações reais.

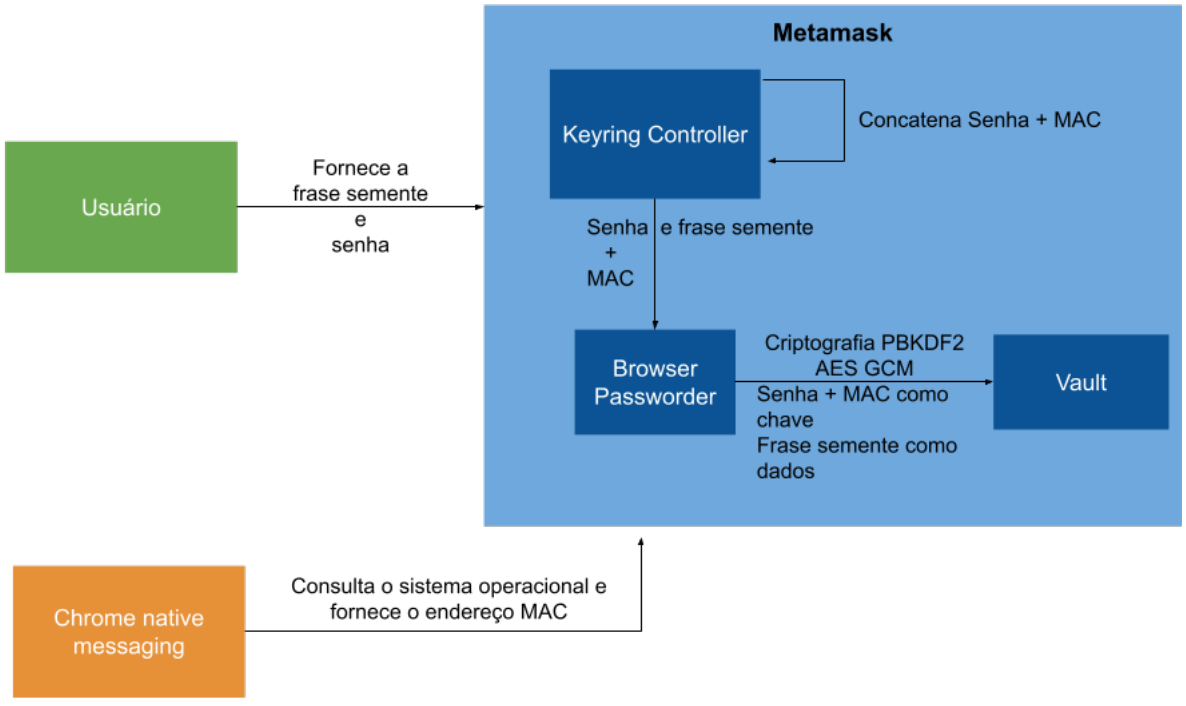
Pode-se observar na Figura 11 o diagrama original da aplicação e o diagrama atualizado proposto acima pode ser observado na Figura 12.

Figura 11 – Diagrama Original



Fonte: Elaborado pelo Autor

Figura 12 – Diagrama do desenvolvimento



Fonte: Elaborado pelo Autor

5 Conclusão

A solução proposta aqui combate ataques de baixa sofisticação, ataques que são feitos por pessoas com baixos conhecimentos sobre as tecnologias utilizadas pela Metamask. A solução também requer que ferramentas já prontas para atacar e roubar o *vault* precisem de atualizações.

Por outro lado, a solução cria inconveniências para os usuários, que ao trocarem de placa mãe ou trocarem a placa de rede em que foi feito o *vault*, terão que entrar novamente com a frase semente para acessar a conta. A Metamask conta com um *decryptor* no repositório de códigos *github* que é feito para caso a aplicação tenha algum tipo de problema, o usuário possa recuperar a frase semente através dele, o *decryptor* deverá ser atualizado para comportar as mudanças feitas.

Utilizando o método de concatenação de senha, o uso de *decryptor* só é viável caso a pessoa tenha acesso ao o endereço MAC, em casos onde o computador sofreu danos ou furto e a pessoa estiver tentando recuperar a frase semente pelo *backup* de um *vault* em armazenamentos *cloud*, esse processo será impossível. O que não é só inconveniência pois também impede que o *vault* seja acessado através de algum *backup* antigo e ignorado pelo usuário que possa estar vulnerável. De toda forma, a frase semente deve ser sempre mantida em segurança e de preferência sem conexão com a internet.

A solução não irá impedir de forma definitiva o acesso do *vault* em computadores que não o gerador, a base da *web* 3 e suas aplicações vem da transparência e da descentralização, seu código aberto requer que a solução seja algo não trivial para realizar esse processo de maneira efetiva.

A Metamask não é vulnerável por assim dizer, para que o *vault* fique vulnerável existem inúmeros fatores que devem acontecer em conjunto. Uma senha fraca é um problema, mas só ela não expõe o *vault*, apenas o acesso ao arquivo do *vault* também não é um problema pois a criptografia AES-GCM PBKDF2 é considerada segura e padrão na indústria. Porém, deve-se haver um esforço para que sempre se construa mais segurança por menores os riscos.

Aos trabalhos futuros e não necessariamente relacionados a Metamask, pode-se explorar a ideia de autenticar um dispositivo com uma senha mestre e depois continuar o uso de uma senha secundária única ao dispositivo. O impacto disso pode ser muito positivo a computação no geral visto os números de ataques envolvendo roubos de credenciais e senhas.

Aos trabalhos futuros relacionados a Metamask, pode-se pensar em usar o *Browser Passworder* para gerar uma criptografia inicial da aplicação utilizando o endereço MAC ou uma outra chave única ao computador, separada da criptografia do *vault*, o mecanismo da

AES-GCM PBKDF2 é eficiente e veloz suficiente para ser realizado duas vezes na aplicação, a primeira para autenticar o computador e a segunda para autenticar o usuário pela senha.

Dessa forma pode-se concluir que a solução proposta de se adicionar um identificador único do equipamento ao processo de criptografia das informações realizado pela Metamask é uma solução a ser considerada para mitigar a falha de segurança existente, porém, impacta na usabilidade da carteira Metamask, podendo gerar dificuldades para o usuário em caso de uma pane de *hardware*. Dessa forma, deve-se considerar as desvantagens apresentadas diante da necessidade atendida e das demais soluções possíveis e seus custos, para assim chegar a uma situação efetivamente segura.

Referências

ANTONOPOULOS, A. M.; WOOD, G. *Mastering ethereum: building smart contracts and dapps*. [S.l.]: O'reilly Media, 2018.

BARTH, A.; JACKSON, C.; REIS, C.; TEAM, T. et al. The security architecture of the chromium browser. In: *Technical report*. [S.l.]: Stanford University, 2008.

BUTERIN, V. Ethereum white paper (2013). 2013. Disponível em: <https://github.com/ethereum/wiki/wiki/White-Paper>. Acesso em: 07/01/2023.

BUTERIN, V. A next-generation smart contract and decentralized application platform. 2014.

DANWAHLIN. *Observable-Store/README.md at main · DanWahlin/Observable-Store*. 2022. Disponível em: <https://github.com/DanWahlin/Observable-Store/blob/main/README.md>. Acesso em: 07/01/2023.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. *IEEE Transactions on Information Theory*, v. 22, n. 6, p. 644–654, 1976.

DWORKIN, M. J. Sha-3 standard: Permutation-based hash and extendable-output functions. Jul 2015. Disponível em: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>. Acesso em: 07/01/2023.

FOXBIT. *O que é blockchain*. 2022. Disponível em: <https://foxbit.com.br/o-que-e-blockchain/>. Acesso em: 07/01/2023.

GLOBENEWSWIRE. *Global Cryptocurrency Market Report 2022-2027 - Industry to Cross a Staggering $32.4 Trillion by 2027, Exploding with a CAGR of 58.4%*. Research and Markets, 2022. Disponível em: <https://www.globenewswire.com/en/news-release/2022/02/25/2392268/28124/en/Global-Cryptocurrency-Market-Report-2022-2027-Industry-to-Cross-a-Staggering-32-4-Trillion-by-2027-Ex
html>. Acesso em: 07/01/2023.

GOOGLE. *Chrome native messaging*. 2014. Disponível em: <https://developer.chrome.com/docs/apps/nativeMessaging/>. Acesso em: 07/01/2023.

GOOGLE. *Google bounty hunters 2023*. 2023. Disponível em: <https://bughunters.google.com/about/rules/5745167867576320/chrome-vulnerability-reward-program-rules>. Acesso em: 07/01/2023.

JOKIć, S.; CVETKOVIć, A. S.; ADAMOVIć, S.; RISTIć, N.; SPALEVIć, P. Comparative analysis of cryptocurrency wallets vs traditional wallets. In: . [S.l.: s.n.], 2019. v. 65.

KHANDE, R.; RAMASWAMI, S.; NAIDU, C.; PATEL, N. An effective mechanism for securing and managing password using aes-256 encryption & pbkdf2. *Technology (IJEET)*, v. 12, n. 5, p. 1–7, 2021.

KODWANI, G.; ARORA, S.; ATREY, P. K. On security of key derivation functions in password-based cryptography. In: *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*. [S.l.: s.n.], 2021. p. 109–114.

LEE, W.-M. *Blockchain Series — How MetaMask Creates Accounts - Level Up Coding*. Level Up Coding, 2021. Disponível em: <https://levelup.gitconnected.com/blockchain-series-how-metamask-creates-accounts-a8971b21a74b>. Acesso em: 07/01/2023.

LENSTRA, A. K.; KLEINJUNG, T.; THOMÉ, E. Universal security. In: *Number theory and cryptography*. [S.l.]: Springer, 2013. p. 121–124.

LUCENA, L. What are metamask "accounts"or "sub-accounts"? and why are they not as private as they are supposed to be? 2022. Disponível em: <https://dev.to/luislucena16/what-are-metamask-accounts-or-sub-accounts-and-why-are-they-not-as-private-as-they-are-supposed-to-be Acesso em: 07/01/2023.

METAMASK. *GitHub - MetaMask/browser-passworder: A simple browserifiable module for encrypting and decrypting JSON-serializable objects with a password.* 2022. Disponível em: <https://github.com/MetaMask/browser-passworder>. Acesso em: 07/01/2023.

METAMASK. *GitHub - MetaMask/KeyringController: A module for managing groups of Ethereum accounts and using them.* 2022. Disponível em: <https://github.com/MetaMask/KeyringController>. Acesso em: 07/01/2023.

METAMASK. 2023. Disponível em: <https://metamask.zendesk.com/hc/en-us/articles/360018766351-How-to-use-the-Vault-Decryptor-with-the-MetaMask-Vault-Data>. Acesso em: 07/01/2023.

NAKAMOTO, S.; BITCOIN, A. A peer-to-peer electronic cash system. v. 4, p. 2, 2008. Disponível em: <https://bitcoin.org/bitcoin.pdf>. Acesso em: 07/01/2023.

NAKOV, S. Practical cryptography for developers (2018). *9786190008705)*, 2018. Disponível em: <https://cryptobook.nakov.com/)>.

PIERRO, M. D. What is the blockchain? *Computing in Science & Engineering*, IEEE, v. 19, n. 5, p. 92–95, 2017.

REGAN, J. *O que é malware? O Guia Definitivo para Malware*. Avg, 2022. Disponível em: <https://www.avg.com/pt/signal/what-is-malware>. Acesso em: 07/01/2023.

RODECK, D.; SCHMIDT, J. What is blockchain. *Forbes Media, Apr*, v. 28, 2022.

SOUZA, R. de. *Brasil sofreu mais de 8,4 bilhões de tentativas de ciberataques em 2020*. 2021. Disponível em: <https://canaltech.com.br/seguranca/brasil-sofreu-mais-de-8-4-bilhoes-de-tentativas-de-ciberataques-em-2020-179493/>. Acesso em: 07/01/2023.

TEAM, C. *Crypto Crime Trends for 2022: Illicit Transaction Activity Reaches All-Time High in Value, All-Time Low in Share of All Cryptocurrency Activity - Chainalysis*. 2022. Disponível em: <https://blog.chainalysis.com/reports/2022-crypto-crime-report-introduction/>. Acesso em: 07/01/2023.

VISCONTI, A.; BOSSI, S.; RAGAB, H.; CALÒ, A. On the weaknesses of pbkdf2. In: SPRINGER. *International Conference on Cryptology and Network Security*. [S.l.], 2015. p. 119–126.

WANG, P. How metamask stores your wallet secret? 2020. Disponível em: <https://www.wispwisp.com/index.php/2020/12/25/how-metamask-stores-your-wallet-secret/>. Acesso em: 07/01/2023.

WASHINGTON, L. C. *Elliptic curves: number theory and cryptography*. [S.l.]: Chapman and Hall/CRC, 2008.

ZHU, W. *How Do Ethereum And Solana Generate Public and Private Keys?* 2022. Disponível em: <https://chainstack.com/how-do-ethereum-and-solana-generate-public-and-private-keys/>. Acesso em: 07/01/2023.