
UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

MATEUS CARACCIOLO MARQUES
(mcm2@cin.ufpe.br)

# GERENCIADORES DE SENHA: SEGURANÇA E USABILIDADE

RECIFE
2022

MATEUS CARACCIOLO MARQUES

# GERENCIADORES DE SENHA: SEGURANÇA E USABILIDADE

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para conclusão do curso de Engenharia da Computação.

**Área de Concentração:** Teoria da Computação

**Orientador:** Ruy José Guerra Barretto de Queiroz

RECIFE
2022

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Marques, Mateus Caracciolo.
Gerenciadores de Senha: Segurança e Usabilidade / Mateus Caracciolo Marques. - Recife, 2023.
40 : il.

Orientador(a): Ruy José Queiroz
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2023.
Inclui referências, anexos.

1. Trabalho de Conclusão de Curso. 2. Trabalho de Graduação. 3. Criptografia. 4. Gerenciadores de Senha. 5. Segurança. I. Queiroz, Ruy José. (Orientação). II. Título.

000 CDD (22.ed.)

## AGRADECIMENTOS

Primeiramente gostaria de começar agradeçendo aos meus familiares, em especial os meus pais, Jairo e Mara Magda, por terem sido um pilar na minha formação pessoal, ética e moral. Pelo suporte financeiro ao longo de toda a minha vida, principalmente nos momentos de pandemia, onde muito foi sacrificado em prol da nossa família. Por serem mais que apenas pais e serem também amigos, compartilhando histórias, gargalhadas e tristezas. Por tudo isto, eu sou grato.

Agradeço a minha companheira, Maria Beatriz, pela presença ao longo de boa parte do meu ensino médio e durante todo o meu percurso acadêmico na faculdade, dando-me forças para continuar nos momentos de recaída, pelos momentos de conforto e carinho, e principalmente por ser uma pessoa maravilhosa, cuja companhia me inspira a ser uma pessoa melhor.

Ao meus amigos, que sempre estiveram comigo nas longas madrugadas para as entregas de projeto e nas vésperas de prova realizando revisões milagrosas. Pelos momentos fora da faculdade, que foram cruciais para manter a compostura e dar forças para sempre seguir em frente. Por serem como irmãos na minha vida e por terem me acolhido como amigo.

Por fim agradeço a instituição UFPE, por providenciar toda a infraestrutura necessária para a minha formação acadêmica e pelas oportunidades apresentadas, e ao corpo docente, que compõe a instituição, pelas aulas e informações para minha formação como engenheiro da computação, principalmente ao meu orientador, Ruy Queiroz, por apresentar-me de forma maravilhosa o tema da criptografia em suas aulas e por ter-me orientado durante a confecção deste trabalho.

## RESUMO

O avanço da internet e a virtualização das necessidades do mundo real torna crucial o uso da criptografia com o propósito de proteger dados sigilosos. Dentre os principais mecanismo para proteção de recursos sigilosos está o uso de senhas. Porém, conforme visto em recentes falhas de segurança, que expuseram milhares de senhas utilizadas na internet, o seu uso é comumente feito de forma incorreta, introduzindo riscos que os atacantes podem se aproveitar a fim de roubar as credenciais do usuário. Uma forma popular de mitigar os problemas gerados pelo uso incorreto das senhas é através de softwares que fazem a administração automática de senhas, de forma a evitar armadilhas comuns no uso rotineiro de senhas. Neste contexto, este trabalho irá realizar uma análise dos protocolos de senhas e seus riscos, além de uma investigação de dois gerenciadores de senhas de código aberto populares: KeePass e Passbolt.



Palavras-Chaves: Senha; Gerenciador de Senhas; Criptografia; Segurança.

## ABSTRACT

The advancement of the internet and the virtualization of real world needs makes it crucial for use of cryptography for the purpose of protecting sensitive data. Among the main mechanism for protecting sensitive resources is the use of passwords. However, in light of recent security breaches, which exposed thousands of passwords used in the internet, its use is commonly done incorrectly, introducing security flaws that attackers can take advantage of in order to steal user credentials. One popular way to mitigate the problems generated by the incorrect use of passwords is through software that manages passwords automatically, in order to avoid common pitfalls in the use of passwords. In this context, this work will carry out an analysis of password protocols and their risks, as well as an investigation of two commonly used open source password managers : KeePass and Passbolt.



Keywords: Password; Password Manager; Cryptography; Security.

## LISTA DE FIGURAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| AES | Advanced Encryption Standard. |
| API | Application Programming Interface. |
| CBC | Cypher Block Chaining. |
| CLI | Command Line Interface. |
| FPGA | Field-Programmable Gate Array. |
| GPU | Graphics Processing Unit. |
| HMAC | Hash-Based Message Authentication Code. |
| HTTPS | Hypertext Transfer Protocol Secure. |
| IP | Internet Protocol. |
| KDF | Key Derivation Function. |
| MVC | Model-View-Controller. |
| NCSC | National Cyber Security Center. |
| NIST | National Institute of Standards and Technology. |
| ORM | Object-Relational Mapping. |
| PBKDF | Password-Based Key Derivation Function. |
| REST | Representational State Transfer. |
| SHA | Secure Hash Algorithm. |
| SMTP | Simple Mail Transfer Protocol. |
| SQL | Structured Query Language. |
| SSL | Secure Socket Layer. |
| TLS | Transport Layer Security. |
| URL | Uniform Resource Locators. |
| UUID | Universally Unique Identifiers. |

SUMÁRIO

# 1 INTRODUÇÃO

## 1.1 CONSIDERAÇÔES INICIAIS

Com o avanço da tecnologia e o surgimento da internet, surgiu a demanda de virtualizar atividades comuns do cotídiano e a criação de novas, tais quais o comércio eletrônico, redes sociais, marketing digital, streaming de vídeos, entre outros. A aderência destes serviços de forma digital acaba trazendo novos desafios. Dentre eles temos a autenticação de usuários, de forma a evitar que usuarios maliciosos consigam realizar uma personificação e consequentemente roubar dados sigilosos do utilizador desses serviços digitais.

Neste contexto, os protocolos de identificação tornam-se indispensáveis para a existência da plataforma digital como conhecemos atualmente. Dentre os principais protocolos de identificação, o mais amplamente utilizado é o de autenticação via senha. Neste o usuário utiliza um login e uma senha, na sua forma mais básica, para realizar autenticação e obter acesso aos seus serviços.

Em muitos sistemas de software, a principal fraqueza na segurança é relacionado ao uso de senhas. Isso é cada vez mais aparente em virtude de recentes falhas de seguranças em diversas empresas, que expõe milhares de senhas armazenadas nos seus bancos de dados do mundo todo, revelando senhas que são consideradas fracas para o padrão NIST de senhas [24, 37]. Estudos realizados em cima de dados vazados, como o realizado pela NCSC [7] e pela Nordpass [12], mostram padrões comumente adotados para as senhas mais comuns. Dentre estes padrões, vemos sequências de números, padrões sequenciais de teclado, nomes populares, músicas famosa, nomes de artistas e personagens fictícios. Além do mais é comum o usuário utilizar a mesma senha para diversos serviços, tornando esses eventuais vazamentos extremamente perigosos.

Diversas alternativas para a autenticação através do uso de senhas foram considerados na literatura, como no caso do trabalho de Bonneau et al, onde é considerado como métodos alternativos o uso de tokens de autenticação, biometria e até mesmo papel para anotar as senhas [16]. Porém, nos dias atuais, é inevitável o uso de senhas para acessar aos principais serviços disponíveis na internet, uma vez que a autenticação através de senha fornece um bom custo benefício entre segurança, usabilidade e custo de implantação para o usuário [17].

Uma alternativa muito eficaz para este problema é através do uso de gerenciadores de senha. Gerenciadores de senha são softwares responsáveis por fazer o gerenciamento de múltiplas senhas através do uso de uma única senha, conhecida como senha mestre. O uso destes softwares aliviam a necessidade de memorização de múltiplas senhas para o usuário e torna mais fácil a aderência de boas práticas na hora de gerar a senha mestre para uso no gerenciador.

Figura 1: Senhas mais comuns utilizadas no Reino Unido em 2019

| Most used in total | Names | Premier League football teams | Musicians | Fictional characters |
|---|---|---|---|---|
| 123456 (23.2m) | ashley (432,276) | liverpool (280,723) | blink182 (285,706) | superman (333,139) |
| 123456789 (7.7m) | michael (425,291) | chelsea (216,677) | 50cent (191,153 | naruto (242,749) |
| qwerty (3.8m) | daniel (368,227) | arsenal (179,095) | eminem (167,983 | tigger (237,290_ |
| password (3.6m) | jessica (324,125) | manutd (59,440) | metallica (140,841) | pokemon (226,947) |
| 111111 (3.1m) | charlie (308,939) | everton (46,619) | slipknot (140,833) | batman (203,116) |

Fonte: Disponível em [7]

## 1.2 JUSTIFICATIVA

Autenticação através de senhas é um dos mecanismos mais utilizados para ter acesso aos serviços mais básicos do dia a dia. Para acessar uma rede social, um serviço de e-mail, uma conta corporativa da empresa e até mesmo o banco, é utilizado senhas. Porém, conforme foi mencionado, há uma péssima adesão à boas práticas de segurança quando se trata de escolher uma senha apropriada para os serviços digitais encontrados na internet. Senhas fáceis de serem adivinhadas e o uso repetido de senhas é algo extremamente comum, visto que é muito comum um usuário ter em média 25 contas distintas em que é necessário o uso de senhas [21].

Boa parte do problema se dá pela incapacidade humana de lembrar senhas de natureza aleatória e de memorizar uma grande quantidade delas. Aderir as políticas de segurança torna-se portanto um processo irreal e demorado, incentivando o usuário a usar de artifícios que muitas vezes não é considerado seguro, como utilizar uma variação da mesma senha ao longo dos serviços, ou adicionar sequências numéricas para tornar a senha mais forte.

Nesse contexto em que se enquadra o presente trabalho, uma vez que é análisado os riscos envolvendo os protocolos de autenticação através de senhas e introduzido o gerenciador de senhas como uma alternativa eficaz que trás consigo a praticidade e segurança necessárias para ser utilizado na prática.

## 1.3 OBJETIVOS

### 1.3.1 **Objetivo Geral**

Apresentar os principais conceitos relacionados com protocolos de identificação através de senhas e o uso de gerenciadores de senhas como alternativa de segurança.

### 1.3.2 **Objetivos Específicos**

1. Expor os principais ataques e vulnerabilidades ao usar senhas fracas e repetidas.
2. Analisar o uso de gerenciadores de senha como mecanismo para autenticação.
3. Compreender as estratégias de segurança adotadas pelos principais gerenciadores de senha open source.

## 1.4 METODOLOGIA

Este trabalho terá como metodologia uma estudo de cunho literário onde será abordado livros teóricos, artigos científicos, revistas científicas e sites a cerca do estado da arte da criptografia utilizada para os protocolos de autenticação através de senhas e dos principais gerenciadores de senha de código aberto.

Em um primeiro momento o foco será a cerca das primitivas criptográficas, os métodos de autenticação através de senha e os principais ataques realizadas em cima destes protocolos. Avaliados os riscos envolvidos nestes protocolos, será feito um estudo de dois gerenciadores de senhas de código aberto, KeePass e Passbolt, através de artigos, revistas e os sites de cada um dos softwares. Por fim considerado as características de usabilidade e segurança fornecidos por cada um destes gerenciadores.

## 1.5 ESTRUTURA DO TRABALHO

Este trabalho está estrutura ao redor de 4 capítulos. No primeiro capítulo é introduzido o contexto no qual o trabalho será apresentado e a sua motivação, além de ser apresentado os objetivos e a metodologia abordada para confeccionamento deste documento. No segundo capítulo é apresentado em mais detalhes a respeito dos protocolos de autenticação através de senhas, sobre como armazenar senhas, os principais tipos ataques comumente realizados nestes esquemas e como se previnir contra os riscos associados. No terceiro capítulo é feito um estudo dos gerenciadores de senha como alternativa de segurança quando se trata do uso de senhas para autenticação. Nele serão apresentado os gerenciadores de senha KeePass e Passbolt e estudado as considerações de segurança e

usabilidade a respeito de cada um deles. Por fim é feita uma análise comparativas entre estes gerenciadores e o método livre de softwares de autenticação de senhas. No quarto e último capítulo é dada as considerações finais a respeito dos tópicos mencionados, onde é exposto os resultados obtidos de forma a orientar a respeito do uso de gerenciadores de senhas e fornecer sugestões para uso pessoal e em equipes.

# 2 PROTOCOLOS DE IDENTIFICAÇÃO COM SENHA

Conforme já foi dito, protocolos de identificação com senha é a forma mais comum de autenticação utilizada em toda a internet, porém, devido a natureza previsível das senhas usualmente escolhidas, torna-se perigoso o seu uso, uma vez que adversários podem realizar ataques eficazes com o intuito de roubar as credenciais do usuário. Portanto, de forma a entender os riscos envolvidos, como se proteger deles e buscar alternativas, é necessário primeiramente entender como funciona um protocólo de identificação com uso de senhas. Boneh e Shoupe [15] definem um protocólo de identificação da seguinte forma:

**Definição 2.1.** *Um **protocólo de identificação** é uma tripla I = (G, P, V) onde:*

- *G é um algoritmo probabilistico para a **geração de chaves**, que não recebe entradas e tem como saída um par (vk, sk), onde vk é chamado de **chave de verificação** e sk é chamado de **chave secreta**.*
- *P é um protocólo interativo chamado de **provador**, que toma como entrada a chave secreta sk gerada por G.*
- *V é um protocolo interativo chamado de **verificador**, que toma como entrada a chave de verificação vk, gerada por G, e tem como saída um estado de* aceitação *ou* negação.

*Nós exigimos então que quando P(sk) interaja com V(vk), V sempre tenha como saída um estado de* aceitação. *Uma protocolo de identificação de senha é quando a chave secreta do provador é igual à **senha** pw.*

## 2.1 ARMAZENAMENTO DE SENHAS

Um aspecto importante da definição do protocólo de identificação é a forma com que é feita a verificação, portanto, não é apenas necessário considerar o uso de senhas complexas para tornar o protocolo seguro, como também é necessário considerar como é feito essa verificação.

Um péssimo exemplo de como pode ser realizado essa verificação é através do armazenamento do par login, senha no banco de dados. Desta forma, para autenticar, basta realizar a comparação do login e senha utilizado. Porém um motivo que torna este método extremamente ineficaz é em casos de invasões de segurança, que por ventura venha a expor o banco de dados, compromentendo todas as senhas dos usuários neste banco.

Uma forma melhor para realizar esta verificação é através do uso de funções de hash criptográficas para o armazenamento de senhas, uma vez que em caso de comprometimento do banco de dados por adversário, as senhas possuem uma camada extra de

proteção através do uso de hashs. Para justificar o uso de funções de hash criptográficas, segue abaixo a sua definição [31]:

**Definição 2.2.** *Uma* ***função de hash criptográfica*** $H : \mathcal{M} \to \mathcal{T}$ *é uma função eficientemente computável que de um espaço de mensagens grande* $\mathcal{M}$ *em um espaço de mensagens curto* $\mathcal{T}$ *com as seguintes propriedades:*

- ***Resistência à pré-imagem****: Dado uma cadeia de n bits* $w$*, deve ser computacionalmente difícil encontrar* $x$ *tal que* $H(x) = w$*.*
- ***Resistência à segunda pré-imagem****: Dado uma entrada* $x$*, deve ser computacionalmente difícil obter um valor* $y$ *tal que* $H(x) = H(y)$*.*
- ***Resistência à colisão****: Deve ser computacionalmente difícil encontrar qualquer par* $(x, y)$ *tal que* $H(x) = H(y)$*.*

Devido as propriedades impostas, caso o banco de dados seja exposto, não é trivial o processo para reverter o hash para obter a senha, como também o processo de forjar uma senha alternativa, cujo valor hash seja o mesmo da senha do usuário, também é inviável.

Para usos em criptografia, a NIST propões um padrão de funções hash consideradas seguras para os diferentes tipos de aplicação na qual é necessária gerar um identificador a partir de uma cadeia arbitrária de caracteres ou digitos binários [19]. É neste documento que é proposto a família SHA de funções hash criptográficas amplamente utilizadas em diversos protocolos de segurança. Abaixo segue uma figura com as características de cada um desses algoritmos.

Figura 2: Família SHA e suas peculiaridades

| Algorithm | Message Size (bits) | Block Size (bits) | Word Size (bits) | Message Digest Size (bits) |
|---|---|---|---|---|
| **SHA-1** | $< 2^{64}$ | 512 | 32 | 160 |
| **SHA-224** | $< 2^{64}$ | 512 | 32 | 224 |
| **SHA-256** | $< 2^{64}$ | 512 | 32 | 256 |
| **SHA-384** | $< 2^{128}$ | 1024 | 64 | 384 |
| **SHA-512** | $< 2^{128}$ | 1024 | 64 | 512 |
| **SHA-512/224** | $< 2^{128}$ | 1024 | 64 | 224 |
| **SHA-512/256** | $< 2^{128}$ | 1024 | 64 | 256 |

Fonte: Disponível em [19]

## 2.2 VULNERABILIDADES EM ESQUEMAS COM SENHAS

Apesar de um protocolo amplamente utilizado, há diversos ataques realizados por adversários que exploram o fato de ser comum o desuso de senhas verdadeiramente aleatórias, ou através de uma má implementação no armazenamento e verificação das senhas.

Isto permite um adversário realizar ataques de força bruta e muitas vezes ataques mais inteligentes, com o uso de uma lista de senhas comumentes utilizadas, para tentar adivinhar a senha do usuário em um processo de tentativa em erro. Se não tiver mecanismo de proteção, tanto na hora de se escolher uma boa senha, como na realização da verificação, o resultado pode ser catastrófico.

Sem contar de outros métodos mais elaborados de engenharia social, como o phishing e suas variantes, onde o usuário é enganado de forma a fornecer as credenciais, normalmente através de adversários que utilizam-se de algum disfarçe para levar o usuário a digitar suas credênciais em ambientes que o atacante possuí controle [36].

## 2.3 ATAQUES DE DICIONÁRIO

Uma categoria de ataques extremamente eficientes na atualidade são os *Ataques de Dicionário*. Ataques de Dicionários são técnicas que utilizam-se da natureza previsível das senhas para montar um dicionário de possíveis senhas onde será realizado a busca, acelerando o processo de tentiva de e erro.

### 2.3.1 **Ataques de Dicionário Online**

Ataques de Dicionário Onlines é a forma mais básica desses tipos de ataque, onde é utilizado um conjunto $\mathcal{D}$ de senhas comuns onde acredita-se que a senha do usuário pertenc. O ataque funciona ao tentar combinações de login com todas as senhas possíveis no dicionário $\mathcal{D}$ até obter uma senha válida. Várias otimizações podem ser feitas nesse ataque, como realizar uma ordenação de senhas por popularidade e até mesmo realizar o ataquede forma paralela, onde vários computadores tentam esse processo de tentativa e erro.

Uma forma de defesa comum é dobrando-se o tempo de resposta a cada duas tentativas erradas de login para um dado login ou endereço IP. Desta forma, após 10 tentativas erradas, o tempo de resposta já é 32 vezes mais lento, dificultando o processo de tentativa e erro. Atacantes, porém, contornam esta estratégia através do uso de uma mesma senha para usuários distintos. Essas tentativas aproveitam-se de diferentes computadores com endereços IP distintos, conhecidos como *bots*, para evitar o mecanismo de duplicação do tempo de resposta.

Muitos métodos não criptográficos, como os CAPTCHAs, são muito bem sucedidos em evitar este tipo de ataque de dicionário [35, 39], porém há uma versão offline desse tipo de ataque que é muito mais difícil de se proteger contra.

### 2.3.2 Ataques de Dicionário Offline

Adversários que são capazes de comprometer servidores utilizados para autenticação e obter o banco de dados utilizado para armazenar informações relacionados com as senhas para verificação, a exemplo de hashs, pode utilizar-se destes dados para realizar uma versão offline do ataque mencionado acima. Importante mencionar que há diversas outras formas de obter estes arquivos de senha sem ser necessário comprometer bancos de dados, por exemplo, através discos rígidos usados, vendidos na internet, que podem conter diversos dados do usuário que não foram propriamente deletados, conforme mostra o estudo de Garfinkle [22].

Como mencionado anteriormente, estes banco de dados possuem encriptação das senhas através do uso de funções de hash. Porém, mesmo com esta camada de proteção adicional, se a senha do usuário estiver dentre as senhas comuns em um certo dicionário $\mathcal{D}$, é possível montar o seguinte ataque de posse da chave de verificação $vk = H(pk)$:

**Algoritmo 1** Ataque de Dicionário Offline

```
for all w ∈ D do
    if H(w) = vk then
        return w
    end if
end for
return fail
```

Ao fim do algoritmo, se a senha $pw$ pertencer ao dicionário, o atacante possuirá $pw$, ou possivelmente $pw'$, onde $H(pw) = H(pw')$.

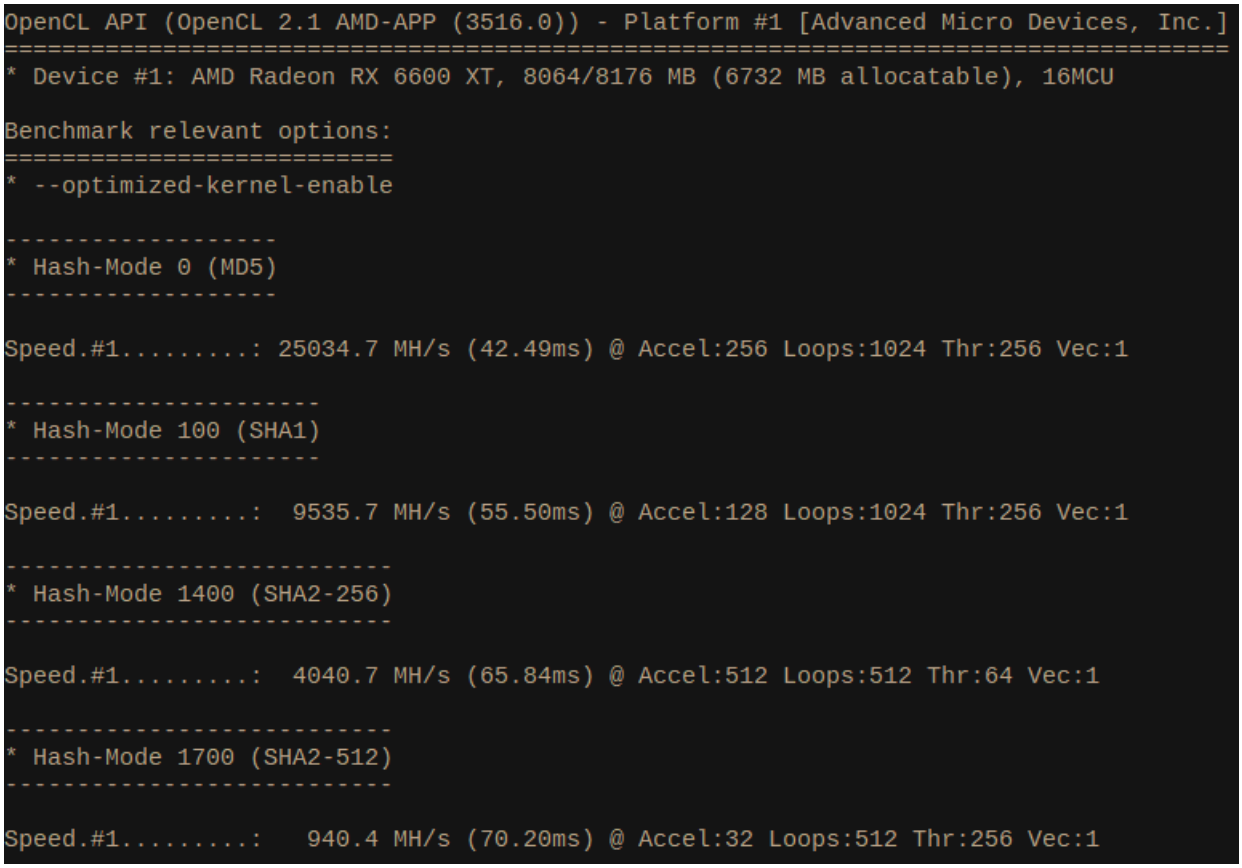
O tempo de execução deste algoritmo é $O(|\mathcal{D}|)$, assumindo-se que o tempo para se computar a função hash é constante. Esse tipo de computação pode ser realizado inteiramente de forma offline, sem necessitar de nenhuma interação entre o provador e o verificador.

O que torna este tipo de ataque tão perigoso em relação a sua versão online é justamente pelo fato de que não é necessário a interação com o servidor, permitindo que o adversário faça suas tentativas em velocidade máxima.

É importante notar que para o sucesso deste ataque, o cálculo da função hash foi considerado constante. Caso o armazenamento de senhas não tenha sido feito de forma a evitar este tipo de ataque, então esta consideração é bem plausível. Funções hash como MD5, SHA-1, SHA-256 e SHA-516 podem ser computadas a uma taxa extremamente alta utilizando uma GPU moderna. Fazendo-se uso de um software especializado em realizar quebras de senhas, como o hashcat [4], em uma GPU como a Radeon RX 6600 XT, pôde-se atingir taxas de 1 bilhão de hashs por segundo para a funções acima mencionadas. Desta forma, se obtivermos algum hash, como o SHA-1, e for utilizado algum dicionário

adequado, a exemplo do disponível pelo CrackStation [1], que possui cerca de 1.5 bilhões de hashs computados previamente, seria possível obter a senha em segundos.

Figura 3: Benchmark do Hashcat em uma Radeon RX 6600 XT




Fonte: De autoria própria.

### 2.3.3 **Ataques de Dicionário Offline com Pré-processamento**

O ataque de dicionário offline discutido acima pode ser tornado ainda mais eficaz para os adversários que desejam obter as credenciais do usuário ao ser realizado uma etapa adicional de pré-processamento do dicionário $\mathcal{D}$ antes de começar o ataque. Dessa forma, ao obter-se a versão hash da senha $vk$, o atacante rapidamente recupera a senha $pw$ se ela estiver contida no dicionário $\mathcal{D}$.

Este novo ataque é divido em duas etapas: a **fase de pré-processamento**, que é realizada antes de se obter qualquer versão hash da senha $vk$, e a **fase de ataque**, onde recupera-se a senha $pw$ através de $vk$.

Assumindo que a função de hash utilizada para o algoritmo na fase de pré-processamento é constante, temos que o tempo de execução dessa fase é $O(|\mathcal{D}|)$. Se a lista $L$ for implementada como uma estrutura de dados que suporta consultas em tempo constante, então a fase de ataque é extremamente rápida e possui tempo de execução $O(1)$.

**Algoritmo 2** Fase de Pré-processamento

1: $L \leftarrow$ Lista Vazia
2: **for all** $w \in \mathcal{D}$ **do**
3: inserir o par $(w, H(w))$ na lista $L$
4: **end for**
5: **return** $L$

**Algoritmo 3** Fase de Ataque

**Require:** Lista $L$
1: **if** $(pw, vk) \in L$ **then**
2: **return** $pw$
3: **else**
4: **return** *fail*
5: **end if**

Uma vez que temos a lista $L$, o atacante pode comprometer diversos hashs de forma eficiente, sem ter que realizar o Algoritmo 1 repetidamente para cada hash. Dessa forma, este ataque pode expor milhares de senhas de usuários através de banco de dados vazados na internet.

Um problema com Algoritmo 2 de pré-processamento utilizado é que requer a construção uma lista $L$ de todos os hashs do dicionário $\mathcal{D}$. Quando $\mathcal{D}$ é o conjunto de todas as senhas de 8 caracteres, por exemplo, temos um total de aproximadamente $2^{52}$ possíveis senhas, tornado a tabela $L$ grande e dificil de armazenar. Uma construção, conhecida como Raibow Tables, permite construir tabelas menores durante a fase de pré-processamento com o custo extra de aumentar a complexidade do algoritmo de ataque. Este tipo de ataque mostra que apenas armazenar hashs da senha não é a forma correta se quisermos proteger as senhas armazenadas.

## 2.4 TORNANDO ATAQUES DE DICIONÁRIO MAIS DIFICEIS

Ataques de Dicionário Offline, pricipalmente com pré-processamento, são uma ameaça real quando se trata de armazenar hashs de senhas fracas no servidor. Portanto é necessário mecanismos extras para tornar mais difícil o processo de quebra de senhas pelo adversário. Discutiremos nesta seção alguns destes mecanismos.

### 2.4.1 **Public Salts**

Uma das formas para evitar o ataque de dicionário offline com pré-processamento é utilizando uma técnica conhecida como **salting** [15]. O salting consiste em adicionar na senha uma cadeia de caracteres aleatória de algum conjunto fixo $\mathcal{S}$ de forma a tornar impraticável a geração e armazenamento das tabelas obtidas na fase de pré-processamento.

Esta cadeia aleatória de caracteres, conhecida como **salt**, é então utilizada para realizar o hash da senha.

Para tornar possível a verificação da senha, é necessário armazenar o salt em puro texto, conforme a figura abaixo.

Figura 4: Armazenamento de senhas com salting

| | | |
|---|---|---|
| $id_1$ | $salt_1$ | $H(pw_1, salt_1)$ |
| $id_2$ | $salt_2$ | $H(pw_2, salt_2)$ |
| $id_3$ | $salt_3$ | $H(pw_3, salt_3)$ |
| $\vdots$ | $\vdots$ | $\vdots$ |

Fonte: Disponível em [15]

Essa nova versão do protocolo funciona da seguinte forma:

- O Algoritmo $G$ gera $pw \leftarrow P$, $salt \xleftarrow{R} \mathcal{S}$, $y \leftarrow H(pw, salt)$;
  tem como saída: $sk := pw$, $vk := (salt, y)$.
- Algoritmo $P$, na entrada $sk = pw$, e o Algoritmo $V$, na entrada $vk = (salt, y)$, interagem da seguinte forma:
  1. $P$ envia $pw$ para $V$;
  2. $V$ emite o estado de *aceitação* se $H(pw, salt) = y$;
     caso contrário emite o estado de *negação*.

Com versão modificada do protocólo de idenfiticação com senha, o adversário possui duas estratégias para atacar hashs de senhas dado um arquivo de senhas $F$:

- A primeira estratégia é adaptar o algoritmo 2 de pré-processamento durante o ataque de dicionário offline. O problema é que agora teremos como possíveis entradas para a função de hash $H$ qualquer elemento do conjunto $\mathcal{D} \times \mathcal{S}$. Portanto isto iria requerer que a lista $L$ gerada tenha um tamanho de $|\mathcal{D}| \times |\mathcal{S}|$, que é muito grande para gerar e consequentemente para armazenar. Portanto o pré-processamento torna-se inviável.
- A segunda estratégia é adaptar o algoritmo 1 de força bruta para executar em cada par $(pw, salt)$ em $\mathcal{D} \times F$. É evidente que neste caso o tempo de execução do algoritmo é $O(|\mathcal{D}| \times |F|)$.

O conjunto $\mathcal{S}$ tem que ser escolhido, portanto, de tal forma que a segunda estratégia seja sempre melhor do que a primeira, mesmo nas situações em que é utilizado a troca entre espaço e tempo no método de raibow table para o pré-processamento de $\mathcal{D} \times \mathcal{S}$. Na prática $\mathcal{S}$ é escolhido de forma que $|\mathcal{S}| = 2^{128}$.

### 2.4.2 **Secret Salts**

O método de salting, apesar defender contra o pré-processamento, não defende contra casos em que a senha de usuário é fraca e seja utilizado o método de força bruta com dicionário offline, como mencionado na seção passada.

Podemos tornar a vida do adversário mais difícil ao adicionar artificialmente entropia na senha [15]. A ideia é utilizar uma cadeia curta de caracteres aleatória, chamada de **secret salt** ou **pepper**, em um conjunto $\mathcal{S}_p$, e incluir na computação da função hash da senha. O detalhe é que dessa vez o secret salt não é exposto em puro texto no arquivo de senhas $F$ utilizado para verificar o hash, conforme a figura abaixo.

Figura 5: Armazenamento de senhas utilizando salt e pepper

| | | |
|---|---|---|
| $id_1$ | $salt_1$ | $H(password_1, salt_1, pepper_1)$ |
| $id_2$ | $salt_2$ | $H(password_2, salt_2, pepper_2)$ |
| $id_3$ | $salt_3$ | $H(password_3, salt_3, pepper_3)$ |
| $\vdots$ | $\vdots$ | $\vdots$ |

Fonte: Disponível em [15]

Com essa modificação, a verificação no protocólo de identificação de senha com secret salt é feito testando-se todas as possibilidades para o secret salt até encontrar um valor cujo hash coincida com o valor no arquivo de senhas $F$. A versão modificada funciona da seguinte forma:

- O Algoritmo $G$ gera $pw \leftarrow P$, $salt \xleftarrow{R} \mathcal{S}$, $pepper \xleftarrow{R} \mathcal{S}_p$, $y \leftarrow H(pw, salt, pepper)$; tem como saída: $sk := pw$, $vk := (salt, y)$.
- Algoritmo $P$, na entrada $sk = pw$, e o Algoritmo $V$, na entrada $vk = (salt, y)$, interagem da seguinte forma:
  1. $P$ envia $pw$ para $V$;
  2. $V$ emite o estado de *aceitação* se $H(pw, salt, p) = y$ para algum $p \in \mathcal{S}_p$; caso contrário emite o estado de *negação*.

Uma escolha típica para o conjunto de peppers é $\mathcal{S}_p = \{0,1\}^{12}$, o que torna o processo de verificação 4096 vezes mais lento do que as versões antigas sem o uso de pepper. Apesar de tornar mais lento o processo de verificação, ainda é rápido o suficiente para o usuário não notar delay durante o processo de autenticação. O mais importante é que agora o adversário deve realizar em média 4096 mais trabalho para encontrar uma senha fraca em um arquivo de senhas.

Uma forma alternativa de se utilizar o secret salt, é o proposto pela NIST [24], onde primeiramente é realizado o hash da senha, utilizando salting, e posteriormente gerado um secret salt, dessa vez em um espaço bem maior ($\mathcal{S}_{\surd} = \{0,1\}^{112}$), para uso em uma função derivadora de chaves. O secret salt desta vez deve ser armazenado, porém em um local distinto de onde é guardado os hashes das senhas, como em um módulo de segurança de hardware [30]. Na próxima seção será discutido melhor sobre funções derivadoras de chaves e seus usos.

O secret salt torna o ataque de dicionário offline mais difícil por que agora o adversário deve realizar a buscar em um espaço $\mathcal{D} \times \mathcal{S}_p$ de possibilidades ao invés de apenas $\mathcal{D}$. Esta técnica aumenta a entropia da senha do usuário sem força-lo a lembrar de uma senha mais complicada.

### 2.4.3 Funções de Hash Lentas

Como visto na técnica de secret salt, tornou-se a tarefa do adversário mais complicada ao tornar o cálculo das funções hash, para verificação, mais lento. Isto ocorre, pois, na maioria das aplicações de criptografia que utilizam-se de funções hash, é esperado que o seu cálculo seja rápido, e, como mencionado para as família de hashs SHA-2, estas funções são extremamente rápidas de se computar, habilitando o adversário a realizar ataques de força bruta de forma muito rápida.

Portanto uma forma alternativa de se proteger uma senha mais fraca é através do uso de funções de hash que são, por design, feitas para ser lentas. Nós iremos agora definir o que consideramos uma função de hash lenta [15].

**Definição 2.3.** *Uma* ***função de derivação de chaves baseado em senha****, ou* ***PBKDF****, é uma função $H$ que tem como entrada uma senha $pw \in \mathcal{P}$, um salt $\in \mathcal{S}$ e um parâmetro de dificuldade $d \in \mathbb{Z}^{\geq 0}$. Esta função tem como saída um valor $y \in \mathcal{Y}$. Nós requerimos que $H$ seja computável por um algoritmo que roda em tempo proporcional a $d$.*

Um método bastante popular para construção de funções hash lentas é a **PBKDF2** [27]. Seja $F$ uma função pseudo-aleatória (PRF) definido em $(\mathcal{P}, \mathcal{X}, \mathcal{X})$, onde $\mathcal{X} := \{0,1\}^n$. A *PBKDF* derivada de $F$, conhecida como $PBKDF2_F$, é definida em $(\mathcal{P}, \mathcal{X}, \mathcal{X})$ e funciona da seguinte forma:

$$PBKDF2_F(pw, salt, d) := \begin{cases} x_0 \leftarrow F(pw, salt) \\ \text{for } i = 1, \ldots, d-1 : \\ \quad x_i \leftarrow F(pw, x_{i-1}) \\ \textbf{return } y \leftarrow x_0 \oplus x_1 \oplus \cdots \oplus x_{d-1} \in \mathcal{X} \end{cases} \tag{1}$$

Na prática, $PBKDF2$ é comumente implementado utilizando a função pseudo-aleatória

HMAC-SHA256 e a dificuldade $d$ é escolhida dependendo da aplicação. Alguns exemplos de uso de tal método é encontrado no IOS 10, no uso de keybags para backup, que são protegidos por $PBKDF2$ utilizando HMAC-SHA256 com $d = 10000$, e na API do Windows 10 para proteção de dados (DPAPI), onde é utilizado $d = 8000$, porém é utilizado o HMAC-SHA512 como função pseudo-aleatória.

### 2.4.4 Funções de Hash com Alto Custo de Memória

Um problema significante com o $PBKDF2$ é que ele é vulnerável a um ataque de hardware paralelo. Este tipo de ataque se dá através do uso do alto paralelismo atingido via hardware para realizar computações da função de hash de forma paralela. Isto é possível pois uma função de hash como a SHA-256 é implementada de forma muito eficiente em hardware, de tal forma que é possível ter varios motores SHA-256 em um hardware dedicado, como uma GPU ou FPGA, capazes de realizem a computação da função hash.

Se, por exemplo, tivermos um chip dedicado com cerca de um milhão de motores SHA-256, sendo que cada um é capaz de computar um milhão de hashs SHA-256 por segundo, então o adversário é capaz de realizar $10^{12}$ computação de hash SHA-256 por segundo. Dessa forma, mesmo se a dificuldade do $PBKDF2$ for colocada em $d = 10000$, um adversário munido de cerca de 500 desses chips seria capaz de realizar uma varredura em todas as senhas de 8 caracteres em menos de um dia.

Isto sugere que, ao invés de utilizarmos funções hashs como a SHA-256, devemos utilizar funções cuja implementação em hardware requer uma grande área em um chip. Uma forma de obter isto é através doo design de funções hash que, para execução, utilizam uma grande quantidade de memoria, necessitando portanto que parte da área para implementação em hardware seja dedicado ao armazenamento necessário para o algoritmo. Funções hashs que necessitam de alto custo de memória são conhecidas como **Memory-Hard Hash Functions**.

Uma construção popular dessas funções hash é a **Scrypt** [34]. Scrypt é construido a partir de uma função hash $h : \mathcal{X} \to \mathcal{X}$, que não necessariamente possui alto custo de memória , onde $\mathcal{X} := \{0,1\}^n$, através de uma primitiva chamada $Scrypt_h$.

Com essa primitiva, nós podemos definir a PBKDF $Scrypt$ através de:

$$Scrypt(pw, salt, d) := \begin{cases} x_0 \leftarrow PBKDF2_F(pw, salt, 1) \\ y \leftarrow Scrypt_h(x_0, d) \\ \textbf{return } PBKDF2_F(pw, y, 1) \end{cases} \tag{2}$$

Na prática, a função pseudo aleatória $F$ utilizada é a HMAC-SHA256 e a função hash $h$ é uma variação da permutação utilizada no algoritmo $Salsa\ 20/8$ [13].

De forma intuitiva, é necessário armazenar $d + 1$ elementos de $\mathcal{X}$ para realizar os

**Algoritmo 4** $Scrypt_h$

**Require:** $x_0 \in \mathcal{X}$, dificuldade $d \in \mathbb{Z}^{>0}$

1: **for** $i = 1, \ldots, d$ **do**
2: $\quad x_i \leftarrow h(x_{i-1})$
3: **end for**
4: $y_0 \leftarrow x_d$
5: **for** $i = 1, \ldots, d$ **do**
6: $\quad j \leftarrow int(y_{i-1}) \mod (d+1)$ // $int(y_{i-1})$ converte $y_{i-1} \in \mathcal{X}$ em um inteiro
7: $\quad y_i \leftarrow h(y_{i-1} \oplus x_j)$
8: **end for**
9: **return** $y_d$

cálculos dos $y_i$ no algoritmo do $Scrypt_h$, uma vez que na linha 6 do Algoritmo 4 obtemos indices em posições supostamente aleatórias para leitura dos valores de $x_j$ nas próximas etapas. Portanto fica de certa forma clara que um algoritmo que executa em tempo $O(d)$ tem de manter o array $(x_0, \ldots, x_d)$ em memória.

O problema surge em possíveis algoritmos que realizam a troca entre espaço e tempo, podendo diminuir o requerimento de memória para execução. Porém é possível demonstrar que, no modelo do oráculo paralelo aleatório, tais algoritmos acabam diminuindo o requerimento de memória de forma proporcional ao aumento no tempo de execução, não tornando o trabalho do adversário mais fácil, mesmo com mais chips que suportam essa versão modificada do algoritmo, uma vez que o algoritmo é mais demanda mais tempo [15].

# 3 GERENCIADORES DE SENHAS

Conforme visto na ultima seção, os métodos de autenticação de senha carregam com sí vários problemas com relação a usabilidade e segurança para o usuário. Em sua maioria, os problemas estão relacionados com a capacidade limitada do usuário gerar senhas verdadeiramente aleatórias e distintas para os diversos serviços que necessitam de autenticação de senha. Neste contexto que surge os gerenciadores de senhas, softwares cuja finalidade é facilitar a autenticação através de senhas, a fim de evitar que o usuário caia nas armadilhas comuns envolvidas nestes protocolos.

## 3.1 CARACTERÍSTICAS DESEJÁVEIS

Uma vez que um gerenciador de senha é um software cujo propósito é facilitar a vida do usuário a se autenticar de forma segura, é necessário que seja fácil de se utilizar e que possua todas as garantias de segurança necessárias para não expor as credênciais do usuário, caso contrário, não possuí valor para quem for utilizar o software.

É comum que haja uma troca quando se trata entre manter alto nível de segurança e usabilidade para o usuário, porém, apesar de parecer contráditório, esses dois aspectos servem um aspecto comum: satisfazer as necessidades do usuário. Segurança restringe o acesso a certas operações indesejadas e evita que erros perigosos ocorram, enquanto que a usabilidade facilita o acesso a operações necessárias e essenciais. É necessário portanto que esses dois fatores sejam considerados no processo de design da aplicação, de forma a evitar conflitos [18, 40].

## 3.2 KEEPASS

KeePass é um gerenciador de senhas grátis, leve e de código aberto destinado a tornar fácil a manutenção das senhas de forma segura para um usuário único [5]. Tem como principal alvo de sistema operacional o Windows, apesar de ter vários ports para outros sistemas operacionais, como o Android, IOS, MacOS e o Linux.

### 3.2.1 **Encriptação da Base de Dados**

O KeePass é um software que roda de forma local, ou seja, ele utiliza uma base de dados com as credenciais do usuário em algum local acessível pelo software, seja na propria memória do computador, como em algum dispositivo de armazenamento removível.

Para utilizar o gerenciador, primeiramente é necessário gerar uma chave mestre, que será usado para o processo de derivação das outras senhas e para encriptação da base de dados. A chave mestre poderá ser uma combinação de vários métodos de autenticação,

como uma senha mestre ou um arquivo chave. A senha mestre é uma cadeia de caracteres, enquanto o arquivo de chave é um arquivo com uma cadeira aleatória de texto gerado na hora da criação. A diferença está no fato que a senha mestre é algo que usuário tem conhecimento, enquanto que o arquivo de senha é algo que o usuário possui. Pode-se utilizar ambos os métodos para autenticação da base de dados, criando desta forma um mecanismo de autenticação de dois fatores.

O KeePass suporta vários algoritmos para encriptação da base de dados do usuário, de forma a torná-lo seguro contra furto. Na versão 2.x, suporta tanto o AES [25] quanto o ChaCha20 [33] como algoritmos para encriptação. O AES é uma cifra de bloco muito popular e que foi adotada pelo governo americano como padrão de encriptação e é utilizada no modo de operação CBC. Já o ChaCha20 é uma cifra de fluxo, sucessora do Salsa20 (que faz parte do eSTREAM Portfolio [2] como um dos algoritmos de cifra de fluxo disponíveis para uso). A integridade e autenticidade do base de dados é garantida em ambos os casos através do uso do HMAC-SHA256 em um paradigma Encrypt-then-MAC [15].

### 3.2.2 Proteção contra Ataques de Dicionário

Uma vez que para seu funcionamento como um gerenciador de senhas é utilizado uma senha mestre para desbloquear a base de dados, é possível que todos os ataque mencionados em esquemas de senhas sejam aplicados, principalmente os ataques de dicionário. Porém o KeePass fornece o uso de algumas funções de derivação de chaves para uso na encriptação da base de dados e para geração das senhas afim de evitar pré-processamento de chaves e adicionar um fator de trabalho que o usuário pode tornar tão grande quanto ele queira de forma a aumentar o esforço computacional de um dicionário e evitar ataques de força bruta.

O KeePass 2.x suporta as seguintes funções de derivação de chave:

- **AES-KDF**: Uma função de derivação de chaves baseada em iterar o AES. A medida que aumenta-se o número de iterações, aumeta-se a complexidade computacional, porém também aumenta o tempo para carregar e salvar o banco de dados.

- **Argon2**: Argon2 é uma nova geração de algoritmos hash de custo de memória alto, vencedor da competição Password Hashing Competition para design de algoritmos que seriam utilizados para realizar o hash de senhas [14]. Conforme mencionado sobre funções hash de alto custo de memória, ela possui uma vantagem em relação ao AES-KDF, uma vez que é resistente a ataques específicos de hardware que exploram a natureza compacta para se implementar algoritmos hash que não de alto custo de memória e explorar o alto grau de paralelismo que esses chips dedicados conseguem realizar.

  A especificação oficial do Argon2 fornece 3 variantes da função de hash: Argon2d,

Argon2id e Argon2i. Destas 3 variantes, O KeePass só fornece suporte para 2 delas: Argon2d e Argon2id. Não é fornecido suporte para o Argon2i, pois este não fornece uma boa segurança contra ataques de hardware dedicados.

KeePass sugere o uso do Argon2d, pois este provê uma melhor resistência contra ataques de hardware paralelo, enquanto o Argon2id possui uma resistência um pouco mais fraco contra estes tipos de ataques, porém possui uma maior resistância contra ataques de canal lateral [6].

zxc

### 3.2.3 **Proteção da Mémoria do Processo**

Enquanto o KeePass está em execução, dados sensíveis são mantidos encriptados na memória do processo. Desta forma, mesmo se for realizado a paginação ou fosse transferido a memória do processo para disco, de forma que fosse possível ser acesso, ainda não seria possível obter informações confidenciais. A exceção se dá quando necessário fazer uso da informação não criptografada na memória do processo. Além disto, por questões de segurança, todo conteúdo sigiloso que não é mais necessário pelo processo é apagado da memória.

### 3.2.4 **Preenchimento Automático**

O Preenchimento Automático é uma feature suportada pelo KeyPass para facilitar o processo de autenticação ao realizar a digitação automática das credenciais armazenadas na base de dados do usuário. No KeePass o processo é feito de forma manual pelo usuário, onde é possível definir sequências de teclas a serem apertados para cada URL a fim de tornar o processo simples para o usuário. Este processo manual é vital para manter seguro a funcionalidade, uma vez que em um artigo de Silver et el. aponta vários perigos associados a esta feature, onde um site supostamente normal pode utilizar de iFrames escondidos e de códigos javascript injetados para extrair senhas auto preenchidas pelos gerenciadores de senha que provêm esta funcionalidade de forma automática.

Também é considerados defesas contra keyloggers, programas criados para capturar as teclas digitadas pelo usuário, uma vez que não há como realizar a distinção entre o pressionar de teclas real e os simulados para o preenchimento automático. A defesa adotada nestes casos é a obfuscação de auto preenchimento de dois canais. A obfuscação utiliza-se da área de transferência para copiar as senhas, de forma que keyloggers só interceptam o pressionar da tecla CTRL + V. É importante notar que a obfuscação não copia e colar toda a senha através da área de transferência, ele faz uma partição da senha, de forma que parte da senha é colocada através da área de transferência e a outra parte

através de teclas simuladas. Isto é feito para evitar aplicativo que espiam a área de transferência.

### 3.2.5 Outras Características

Outras características que são desejáveis implementadas pelo KeyPass é com respeito a portabilidade. Uma vez que que a base de dados é um simples arquivo encriptado e o arquivo de senha é armazenável em qualquer lugar, é possível utilizá-lo em qualquer ambiente que possui uma instância do KeePass de forma simples, utilizando, por exemplo, um USB ou o drive para compartilhar os arquivos. Outro ponto interessante é que ele possui uma versão portável para download, onde não é necessário permissões para baixar. Desta forma pode-se utilizar em ambientes públicos, onde não há permissão para realizar downloads de executáveis.

O KeePass também fornece um ótimo gerador de senhas aleatórios, onde é possível realizar a customização de senhas para atender políticas de seguranças normalmente impostas por contas corporativas. Por fim, possui uma integração com plugins extremamente acessível, fornecedo facilidades para realizar algumas tarefas mais tediosas, como o backup dos arquivos de senha.

## 3.3 PASSBOLT

Passbolt [8], como o KeePass, é um gerenciador de senhas de código aberto, porém o que torna-o diferente é que foi, por design, feito para o uso colaborativo entre equipes dentro de uma organização. Portanto oferece diversos mecanismos para compartilhamento e gerenciamento de credenciais de uma corporação.

### 3.3.1 Arquitetura da Aplicação

Uma vez que o Passbolt tem como objetivo o gerenciamento de credenciais de um time, sua arquitetura difere radicalmente da adotada pelo KeePass, já que o KeePass, em sua essência, é feito para o uso pessoal. Passbolt oferece uma arquitetura de cliente e servidor, uma vez que é necessário um ambiente centralizado para armazenar as credenciais que serão compartilhadas pela equipe. Possui suporte para instalação em diversos servidores amplamente utilizados pela comunidade, como o Docker, Kubernetes, servidores Linux e para provedores de Cloud como o AWS e o Digital Ocean.

O servidor utiliza-se de uma arquitetura MVC para fornecer respostas das requisições realizadas pelo cliente através do uso de HTTPS de forma REST. O servidor depende de uma instância do MariaDB e de um sistema de Cache, que pode ser ou o sistema de arquivos local ou o Redis, para responder as requisições da aplicação.

Uma vez que a segurança é a principal preocupação de um gerenciador de senhas, o Passbolt utiliza encriptação de ponta a ponta e o processo de encriptar e decriptar é sempre feito pelo lado do cliente. O servidor é principalmente usado para manter a integridade e armazenamento dos dados que serão compartilhados pela equipe.

A aplicação costuma ser uma extensão de browser ou uma CLI, onde é feito o acesso ao servidor e a interação com o usuário para adicionar e compartilhas senhas, além de fornecer métodos para copiar as credenciais para a área de transferência e preenchimento automático de formulários.

É através da aplicação que é realizado boa parte do processo de configuração do servidor em um contato inicial, como configurar o servidor SMTP utilizado para enviar notificações de e-mail, configurar o certificado SSL para uso em seções HTTPS e na geração dos pares de chave pública e privada para autenticação e gerenciamento do banco de dados. Mais detalhes de como é o processo utilizado para garantir segurança no Passbolt é apresentado nas próximas seções.

### 3.3.2 **Considerações Criptográficas**

O Passbolt, diferentemente de outros gerenciadores de senha, não utiliza-se de encriptação de chave simétrica compartilhada entre os usuários. Em vez disto, é usado extensivamente criptografia de chave pública, mais especificamente o OpenPGP para encriptar os dados. Este tipo de esquema é desejado, pois facilita o processo de compartilhamento de credenciais entre uma equipe, conforme a figura abaixo.

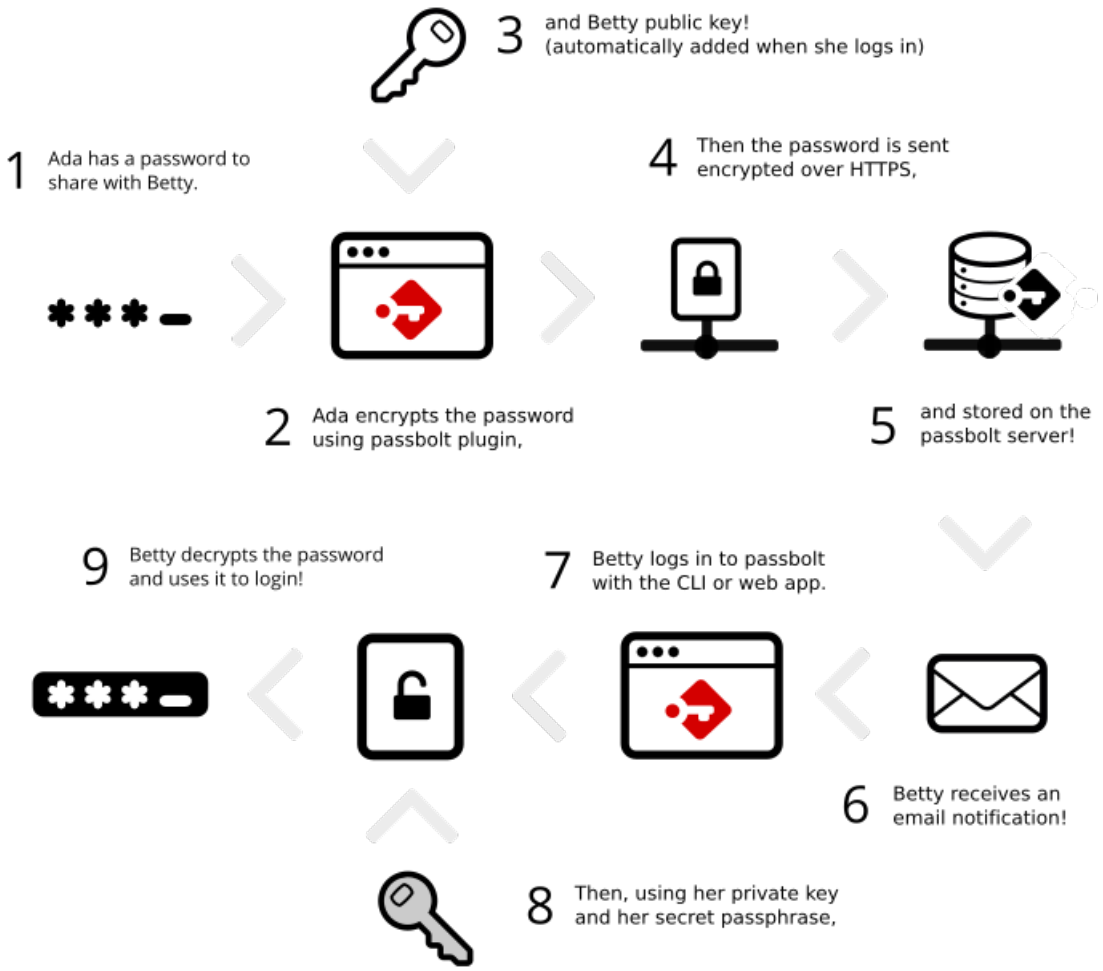
Para fins de encriptação de chave pública, o par de chave pública e privada é gerada na hora da criação do usuário. Também é utilizada uma frase secreta de forma a tornar o processo mais robusto ao habilitar autenticação de dois fatores, posse e conhecimento. A chave privada é encriptada, utilizando-se a frase secreta, e mantida no armazenamento do cliente. Já as chaves públicas são sincronizadas no servidor, onde é utilizado um protocolo de autenticação para realizar a busca ou atualização de chaves públicas.

### 3.3.3 **Armazenamento dos Dados**

O armazenamento de dados que não são considerados sigilosos, como o nomes, AES e comentários, é feito em puro texto, livre de encriptação, tanto no cliente quanto no servidor. Já informações sigilosas, como senhas ou descrições, são mantidos encriptados e podem ser feitos disponíveis quando necessário em puro texto, como em casos de cópia para área de transferência. Porém nunca serão armazenados em puro texto no sistema de arquivos tanto do cliente quanto do servidor.

Para dados em trânsito, é realizado encriptação através o TLS, um protocólo de segurança em cima da camada de transporte. Neste nível, a segurança não é controlada pelo

Figura 6: Compartilhamento de senha utilizando criptografia de chave pública




Fonte: Disponível em [10]

Passbolt. Quem define o nível de segurança é a organização responsável pelos certificados SSL e pela configuração do servidor web escolhido pelo provedor de hospedagem.

### 3.3.4 **Gerenciamento de Chaves**

Conforme mencionado, o Passbolt faz amplo uso de pares de chaves públicas e privadas, tanto para o servidor, quanto para o cliente, em seus protocolos.

A chave OpenPGP do servidor é gerada durante a sua configuração. Também pode ser gerada através de ferramentas manuais do administrador que está realizando a hospedagem do servidor. Esta chave é usada majoritariamente no mecanismo de autenticação com o servidor. O cliente baixa a chave do servidor durante a configuração da conta e armazena na configurações da extensão web utilizada para rodar o Passbolt.

Já a chave do cliente é gerada na hora da criação da conta. Por padrão é utilizado chaves RSA de 2048 bits, porém é possível importar chaves RSA maiores. Até este

momento, não é suportado chaves baseadas em criptografia de curvas elipticas. Durante a configuração também é exigido a criação de uma frase secreta aderindo a certos padrões de complexidade. A chave secreta é então encriptada por meio da frase secreta. Todo o armazenamento das chaves secretas, como das configurações, é feita no armazenamento local da extensão web.

De forma a tornar o processo de troca de chaves mais simples para o usuário e evitar o gerenciador de chaves, Passbolt introduz uma autoridade central responsável por fornecer endpoints para distribuir as senhas entre os usuários. Desta forma é possível fazer buscas que retornam as chaves OpenPGP armazenadas no servidor com um parâmetro adicional de só retornar as chaves modificadas até um certo intervalo de tempo. Por padrão, a extensão web armazena esses intervalos de tempos para evitar reimportar chaves de usuário marcados como não alterados no servidor.

### 3.3.5 **Autenticação**

GPGAuth é um protocolo que utiliza o padrão OpenPGP para autenticar os usuários. É baseado na implementação do OpenPGP conhecido como GnuPG ou GPG [3]. Ele funciona através de um desafio que é gerado pelo servidor que deve ser resolvido pelo cliente de forma a comprovar a sua autenticidade.

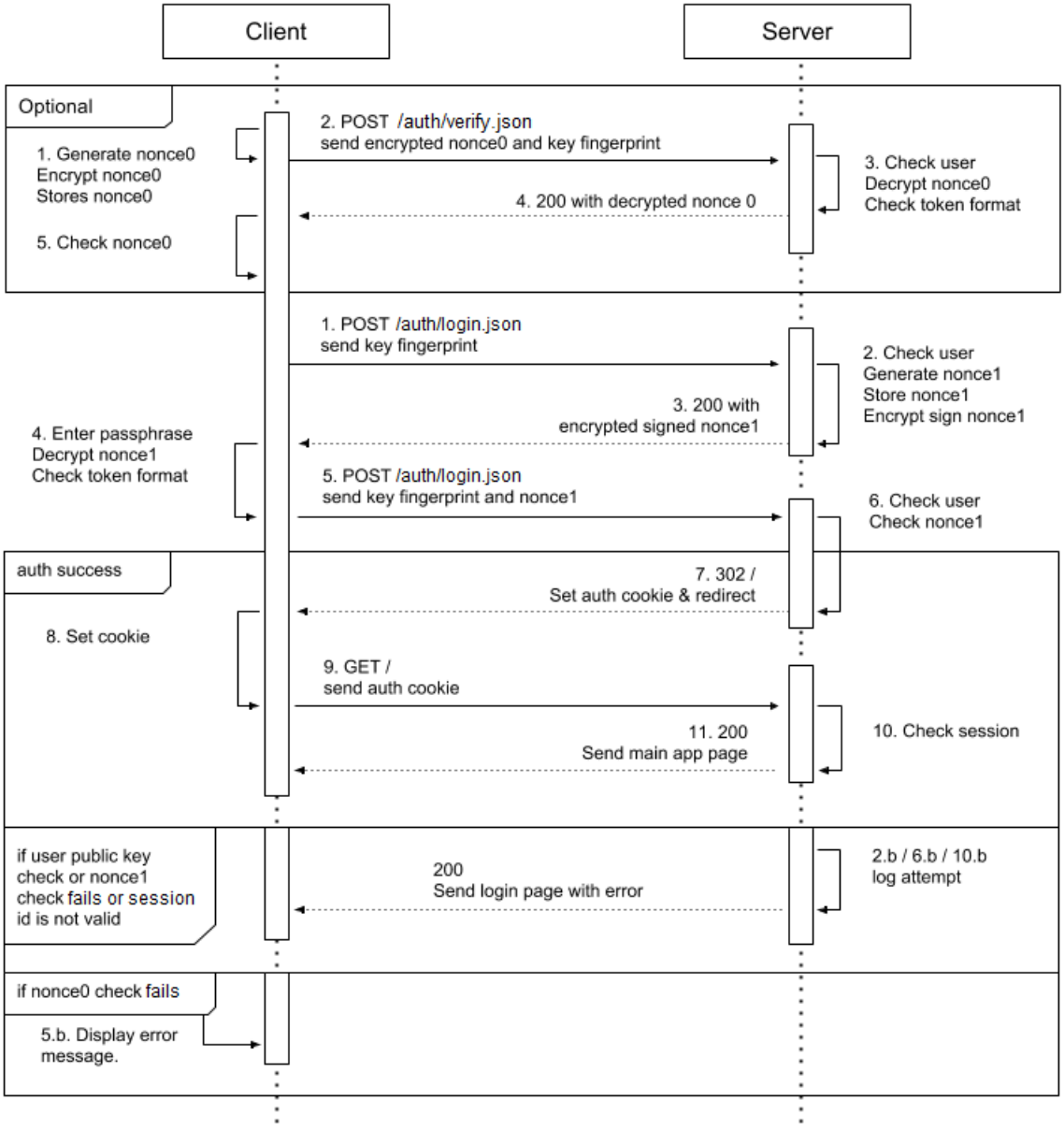
Este desafio é gerado através da chave pública do usuário que quer se autenticar. O servidor checa a existência dessa chave pública no seu banco de dados, que é gerada para um usuário cadastrado no momento da sua criação, e utiliza-se dessa chave para encriptar um valor aleatório e enviar de volta para o cliente. O cliente recebe este desafio e deve decriptar com sua chave privada o token recebido e enviar de volta para o servidor, juntamente com sua chave pública. O servidor então checa se o valor decriptado é o mesmo do desafio gerado. Caso sim, estabelece-se uma seção e a autenticação está pronta.

Neste protocolo, pode-se envolver uma etapa adicional para verificação da identidade do sevidor, para casos de apreensão do domínio do servidor. Este processo de verificação é essencialmente o processo reverso de autenticação. Desta vez é o usuário que gera um desafio para o servidor utilizando-se de sua chave pública, que é obtida através de um endpoint que a expõe, e ele que responde com o valor decriptado do token enviado pelo cliente. Se os valores coincidirem, então pode-se verificar a identidade do servidor. Deve-se notar que este não é um processo de autenticação de ponta a ponta do servidor, pois não protege de ataques man-in-the-middle.

Este processo de autenticação tem algumas vantagens em relação ao método usual baseado em um formulário. Primeiramente é possível se defender de phishing, onde um adversário consegue fazer o usuário digitar sua frase secreta em um formulário malicioso de forma a obter a frase secreta, pois com apenas esta informação não é possível derivar a chave privada do usuário, que é mantida no sistema de arquivos encriptada através

da chave secreta. Também fornece uma segurança maior, pois o token utilizado para autenticação é gerado aleatoriamente para cada tentativa de autenticação e possui um tamanho de 122 bits, que é mais forte que uma senha comum. E por fim, o token utilizado para o desafio é independente da frase secreta utilizada pelo usuário. Então em um eventual vazamento, não há como deduzir a frase secreta do usuário.

Figura 7: Etapas de autenticação utilizando o GPGAuth




Fonte: Disponível em [9]

### 3.3.6 **Autorização**

Uma vez que o Passbolt é destinado a ser utilizado em equipes, há diferentes tipos de papeis atribuidos ao usuário de forma a permitir o gerenciamento dos times. Na hierarquia

de papeis, temos os usuários administradores e usuários normais. O usuário administrador é quem possui acesso para modificar a instancia do servidor e atribuir outros papeis à nivel de grupo. É ele o responsável por configurar as notificações de e-mail, os mecanismo de autenticação de multiplos fatores e da criação e destruição de usuários e grupos. Dentro de um grupo, o gerente o responsável por adicionar ou remover novos usuários. Há também a hierarquias de recursos, limitando o acesso de quem pode acessar, modificar e deletar.

Desta forma há todo um sistema hierarquico que permite de forma simples e prática a adoção do gerenciador de senhas entre um time, sendo possível compartilhar senhas e garantir permissões para modificar e deletar credenciais.

### 3.3.7 Mitigação de Riscos

Passbolt implementa vários mecanismos a fim de evitar ataques comums em aplicações que rodam em browsers. Alguns exemplos de ataques comuns que são evitados são: Phishing, Cross Site Scripting, SQL Injection e Cross Site Request Forgery.

**Phishing**

Nesse tipo de ataque, o adversário cria uma página de login semelhante a utilizada pela extensão web ou injeta um código javascript em uma legitima página do Passbolt. Passbolt faz uso de tokens de segurança para evitar estes problemas. Os tokens de segurança funcionam como um conjunto de 3 caracteres escolhidos previamente e uma cor, dessa forma forjar uma página com um formulário se torna impraticável. Para previnir formulário invisíveis por cima dos formulários legítimos, é utilizado um evento de foco que faz com que haja a mudança de cor do token de segurança.

Figura 8: Uso de tokens de segurança para evitar phishing


Fonte: Disponível em [9]

**Cross Site Scripting**

Estes ataques utilizam da ausência de sanitização de texto em páginas web para introduzir códigos javascript maliciosos para obter as credenciais do usuário. Passbolt lida com isto no lado do cliente ao utilizar recursos de escapes apropriados fornecidos pelas bibliotecas (React). Adicionalmente, Passbolt utiliza políticas de segurança para previnir rodar códigos javascript inline ou incluir javascript de domínios de terceiros.

**SQL Injection**

Injeções de SQL é quando utiliza-se da linguagem SQL para introduzir expressões que são sempre verdadeiras e executar buscas especificadas pelo adversário que está injetando este SQL. Passbolt evita este tipo de problema ao utilizar uma camada de acesso de dados conhecida como ORM. Por padrão, esta camada evita boa parte das injeções de SQL. Porém inputs de usuários são sempre validados de forma a evitar que seja possível realizar uma injeção em cima desta nova camada de acesso ao banco de dados.

**Cross Site Request Forgery**

Cross-Site Request Forgery é um ataque que força um usuário a executar ações indesejadas em um aplicativo da web no qual ele está autenticado no momento, podendo acionar modificações de recursos em um site através de um formulário que foi enviado através de um outro domínio.

Um dos mecanismos utilizados para evitar estes ataques é através de um UUID, que é utilizado para identificar todas as entidades, necessário para todas as operações DELETE e PUT, tornando os URLs difíceis de adivinhar para um atacante.

Contra o restante dos risco é utilizado o Middleware PSR-7 em operações POST, PUT e DELETE. Ele funciona definindo um csrfToken no cookie e em um campo de entrada oculto nos formulários. A ficha será enviado junto com o cookie e como parte dos dados da solicitação. Como alternativa para o Ajax, os tokens podem ser enviados por meio de um Cabeçalho X-CSRF-Token. O componente de middleware irá comparar os dados solicitados e o valor do cookie e se os dados estiverem ausentes ou os valores incompatíveis, um erro será retornado.

### 3.3.8 Vulnerabilidades

Apesar de se proteger da grande maioria dos ataques mais famosos em aplicações web, ainda há riscos envolvidos caso o adversário consiga ter acesso a máquina do usuário. O passbolt não fornece nenhum tipo de segurança contra keyloggers, softwares que tem acesso ao sistema de arquivos, aplicações que podem observar a área de transferência e de adversários capazes de lerem o conteúdo em memória do processo que está em execução no browser.

# 4 CONSIDERAÇÔES FINAIS

Conforme visto ao longo do trabalho, protocolos de identificação através de senhas é a forma mais popular de autenticação ao longo da internet. Porém, a sua efetividade é diretamente ligado ao quão complexo é a senha e, em ocasiões de eventuais invasões de segurança, na capacidade de gerar senhas distintas. Isto causa uma sobrecarga cognitiva para o usuário que deseja agilidade no uso da internet do cotidiano, levando ao uso de senhas consideras fracas para os padrões de segurança da atualidade. Isto torna as credenciais dos usuários expostas aos diversos ataques de força bruta mencionados no trabalho.

Gerenciadores de senhas surgem nesse contexto como uma alternativa para aliviar esta sobrecarga do usuário e fornecer uma interface de interação de fácil uso e com características de seguranças mais robustas. Dentre os gerenciadores de senhas disponíveis, foram considerados o estudo de dois deles: KeePass e o Passbolt. A escolha destes dois se deve ao fato de serem ambos de código aberto e por possuirem públicos alvos distintos. O KeePass é uma alternativa para usuários que desejam gerenciar suas senhas de maneira individual, enquanto o Passbolt é uma alternativa para equipes.

É portanto recomendado o uso de gerenciadores de senhas sempre que possível, principalmente algum simples como o KeePass. Caso seja necessário utilizar um gerenciador mais complexo e escalável, aconselha-se o Passbolt. Porém é importante notar que o processo de configuração é mais complexo e para utilizá-lo de forma eficaz é necessário entender alguns dos mecanismos utilizados por ele para evitar problemas de segurança.

## REFERÊNCIAS


[1] CrackStation. Disponível em: https://crackstation.net/. (Acesso em Março de 2023).

[2] eSTREAM: the ECRYPT Stream Cipher Project. Disponível em: https://www.ecrypt.eu.org/stream/. (Acesso em Fevereiro de 2023).

[3] GnuPG. Disponível em: https://gnupg.org/. (Acesso em Março de 2023).

[4] Hashcat. Disponível em: https://hashcat.net/hashcat/. (Acesso em Março de 2023).

[5] KeePass Password Safe. Disponível em: https://keepass.info/. (Acesso em Fevereiro de 2023).

[6] KeePass Security. Disponível em: https://keepass.info/help/base/security.html. (Acesso em Fevereiro de 2023).

[7] Most hacked passwords revealed as UK cyber survey exposes gaps in online security. Disponível em: https://www.ncsc.gov.uk/news/most-hacked-passwords-revealed-as-uk-cyber-survey-exposes-gaps-in-online-security. (Acesso em Janeiro de 2023).

[8] Passbolt. Disponível em: https://www.passbolt.com/credits. (Acesso em Março de 2023).

[9] Passbolt - Security White Paper. Disponível em: https://www.passbolt.com/credits. (Acesso em Março de 2023).

[10] Passbolt api documentation. Disponível em: https://help.passbolt.com/api. (Acesso em Março de 2023).

[11] Side-channel attacks in web browsers: practical, low-cost, and highly scalable. Disponível em: https://www.cs.columbia.edu/2015/spy-in-the-sandbox/. (Acesso em Fevereiro de 2023).

[12] Top 200 Most Common Passwords. Disponível em: https://nordpass.com/most-common-passwords-list/. (Acesso em Janeiro de 2023).

[13] Bernstein, D. J. The salsa20 family of stream ciphers. *New stream cipher designs: the eSTREAM finalists* (2008), 84–97.

[14] Biryukov, A., Dinu, D., and Khovratovich, D. Argon2: new generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (2016), IEEE, pp. 292–302.

[15] Boneh, D., and Shoup, V. *A Graduate Course in Applied Cryptography*. 2023. Disponível em: http://toc.cryptobook.us/.

[16] Bonneau, J., Herley, C., Van Oorschot, P. C., and Stajano, F. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE symposium on security and privacy* (2012), IEEE, pp. 553–567.

[17] Chaudhary, S., Schafeitel-Tähtinen, T., Helenius, M., and Berki, E. Usability, security and trust in password managers: A quest for user-centric properties and features. *Computer Science Review 33* (2019), 69–90.

[18] Cranor, L. F., and Garfinkel, S. Guest editors' introduction: secure or usable? *IEEE security & privacy 2*, 5 (2004), 16–18.

[19] Dang, Q. H. Secure hash standard.

[20] Finney, H., Donnerhacke, L., Callas, J., Thayer, R. L., and Shaw, D. OpenPGP Message Format. RFC 4880, Nov. 2007.

[21] Florencio, D., and Herley, C. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web* (2007), pp. 657–666.

[22] Garfinkle, S., and Shelat, A. Remembrance of data passed: A study of disk sanitization practices. *IEEE Security & Privacy 1*, 1 (2003), 17–27.

[23] Gaw, S., and Felten, E. W. Password management strategies for online accounts. In *Proceedings of the second symposium on Usable privacy and security* (2006), pp. 44–55.

[24] Grassi, P. A., Fenton, J. L., Newton, E., Perlner, R., Regenscheid, A., Burr, W., Richer, J., Lefkovitz, N., Danker, J., Choong, Y.-Y., et al. Nist special publication 800-63b: digital identity guidelines. *National Institute of Standards and Technology (NIST)* (2017).

[25] Heron, S. Advanced encryption standard (aes). *Network Security 2009*, 12 (2009), 8–12.

[26] Hranickỳ, R., Zobal, L., Ryšavỳ, O., and Kolář, D. Distributed password cracking with boinc and hashcat. *Digital Investigation 30* (2019), 161–172.

[27] Kaliski, B., and Rusch, A. Rfc 8018: Pkcs# 5: Password-based cryptography specification version 2.1, 2017.

[28] Li, L., Berki, E., Helenius, M., and Savola, R. New usability metrics for authentication mechanisms. In *Proceedings of SQM and INSPIRE 2012, August 21-23, 2012, Tampere, Finland*. 2012, pp. 240–250.

[29] Luevanos, C., Elizarraras, J., Hirschi, K., and Yeh, J.-h. Analysis on the security and use of password managers. In *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)* (2017), IEEE, pp. 17–24.

[30] Mavrovouniotis, S., and Ganley, M. Hardware security modules. In *Secure Smart Embedded Devices, Platforms and Applications*. Springer, 2013, pp. 383–405.

[31] Mironov, I., et al. Hash functions: Theory, attacks, and applications. *Microsoft Research, Silicon Valley Campus* (2005), 1–22.

[32] Morris, R., and Thompson, K. Password security: A case history. *Communications of the ACM 22*, 11 (1979), 594–597.

[33] Nir, Y., and Langley, A. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, May 2015.

[34] Percival, C., and Josefsson, S. The scrypt password-based key derivation function. Tech. rep., 2016.

[35] Pinkas, B., and Sander, T. Securing passwords against dictionary attacks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (2002), pp. 161–170.

[36] Salahdine, F., and Kaabouch, N. Social engineering attacks: A survey. *Future Internet 11*, 4 (2019), 89.

[37] Scarfone, K., and Souppaya, M. Guide to enterprise password management (draft). *NIST special publication 800*, 118 (2009), 800–118.

[38] Silver, D., Jana, S., Boneh, D., Chen, E., and Jackson, C. Password managers: Attacks and defenses. In *23rd USENIX Security Symposium (USENIX Security 14)* (2014), pp. 449–464.

[39] Von Ahn, L., Blum, M., Hopper, N. J., and Langford, J. Captcha: Using hard ai problems for security. In *Eurocrypt* (2003), vol. 2656, Springer, pp. 294–311.

[40] Yee, K.-P. Aligning security and usability. *IEEE Security & Privacy 2*, 5 (2004), 48–55.