ADRIELI FERNANDA DE MORAES SUAREZ

ALAN ROCHA ZAN

ANÁLISE FORENSE DIGITAL COM FOCO EM MANIPULAÇÃO DE DADOS

CAMPINAS
2025

ADRIELI FERNANDA DE MORAES SUAREZ

ALAN ROCHA ZAN

ANÁLISE FORENSE DIGITAL COM FOCO EM MANIPULAÇÃO DE DADOS

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do diploma do Curso Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - Câmpus Campinas

Orientador: Prof. Dr. Diego Fernandes Goncalves Martins

Campinas
2025

FICHA CATALOGRÁFICA
Biblioteca IFSP – Campus Campinas
Tatiane Helena Borges de Salles
CRB8/8946

Suarez, Adrieli Fernanda de Moraes..

Análise forense digital com foco em manipulação de dados : / Adrieli Fernanda de Moraes Suarez; Alan Rocha Zan. – 2025.
51 f. il.
Trabalho de Conclusão de Curso (Graduação em Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia do Estado de São Paulo, Campinas, SP, 2025 .
Orientador(a): Diego Fernandes Gonçalves Martins.

1. Crime por computador - investigação. 2. Hackers. 3. Lógica fuzzy. . . I. Orientador(a) Diego Fernandes Gonçalves Martins. II. Instituto Federal de Educação, Ciência e Tecnologia de São Paulo. III. Título.

ATA N.º 18/2025 - TADS-CMP/DAE-CMP/DRG/CMP/IFSP

Ata de Defesa de Trabalho de Conclusão de Curso - Graduação

Na presente data, realizou-se a sessão pública de defesa do Trabalho de Conclusão de Curso intituladoAnálise Forense Digital com Foco em Manipulação de Dados apresentado(a) pelo(a) aluno(a) Alan Rocha Zan (CP3025039) e Adrieli Fernanda de Moraes Suarez (CP3025446)do Curso SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS (Campus Campinas). Os trabalhos foram iniciados às 18h00min pelo(a) Professor(a) presidente da banca examinadora, constituída pelos seguintes membros:

Membros	IES	Presença (Sim/Não)
Diego Fernandes Gonçalves Martins (Presidente/Orientador)	IFSP	SIM
Fábio Feliciano de Oliveira (Examinador 1)	IFSP	SIM
Marcos Brandão Rios (Examinador 2)	IFSP	SIM

Observações:

A banca examinadora, tendo terminado a apresentação do conteúdo da monografia, passou à arguição do(a) candidato(a). Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelo(a) aluno(a), tendo sido atribuído o seguinte resultado:

[x] Aprovado(a)

[ ] Reprovado(a)

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu lavrei a presente ata que assino juntamente com os demais membros da banca examinadora.

Campus Campinas, 24 de outubro de 2025

Documento assinado eletronicamente por:

- Diego Fernandes Goncalves Martins, PROFESSOR ENS BASICO TECN TECNOLOGICO , em 24/10/2025 16:08:13.
- Fabio Feliciano de Oliveira, PROFESSOR ENS BASICO TECN TECNOLOGICO , em 24/10/2025 16:09:20.
- Marcos Brandao Rios, PROF ENS BAS TEC TECNOLOGICO-SUBSTITUTO, em 27/10/2025 16:05:09.

Este documento foi emitido pelo SUAP em 21/10/2025. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 1050422
Código de Autenticação: 7c6a3e3e4c

RESUMO

Atualmente as investigações forenses na área digital têm de lidar com grandes volumes de dados, tornando sua análise manual inviável. Este trabalho visa à compreensão do funcionamento dos métodos de análise digital forense com foco na manipulação de dados. A computação forense é uma área da ciência que tem como principal objetivo descobrir a autoria de crimes ligados à área de informática, podendo ser chamado também de crimes cibernéticos. Para o conhecimento necessário à elaboração deste projeto, foram estudados assuntos de forense computacional, bem como suas implicações legais, preservação, análise e apresentação. Além disso, foram analisadas as dificuldades enfrentadas durante o processo de manipulação dos dados de uma forma geral, considerando os desafios técnicos e legais envolvidos. O estudo aborda em específico a técnica Hash Fuzzy, também conhecida como hash de similaridade, que permite identificar alterações parciais em arquivos digitais. No que se refere aos crimes cibernéticos, o trabalho discute sua crescente complexidade e variedade, abordando desde invasões de sistemas, roubo de dados, até crimes financeiros e ataques de ransomware. Esses crimes, muitas vezes praticados de forma remota e anônima, dificultam a identificação dos responsáveis e exigem técnicas avançadas de investigação digital. O estudo evidencia, portanto, a importância da integração entre conhecimentos técnicos e jurídicos para que as investigações sejam eficazes e respeitem os princípios legais, garantindo a validade das provas digitais coletadas no combate aos crimes cibernéticos. Dessa forma, o trabalho contribui para o entendimento das práticas e limitações no cenário atual da computação forense aplicada a mídias criptografadas, fornecendo subsídios teóricos para a atuação em casos reais de crimes cibernéticos.

Palavras chave: computação forense; crimes cibernéticos; hash fuzzy, manipulação de dados; provas digitais.

ABSTRACT

Currently, forensic investigations in the digital area have to deal with large volumes of data, making manual analysis unfeasible. This work aims to understand how digital forensic analysis methods work, focusing on data manipulation. Computer forensics is an area of forensic science whose main objective is to discover the authorship of crimes related to the computer area, which can also be called cybercrimes. In order to obtain the knowledge necessary for the preparation of this project, computer forensics issues were studied, as well as their legal implications, preservation, analysis and presentation. In addition, the difficulties faced during the data manipulation process in general were analyzed, considering the technical and legal challenges involved. The study specifically addresses the Fuzzy Hash technique, also known as similarity hash, which allows the identification of partial changes in digital files. Regarding cybercrimes, the work discusses their increasing complexity and variety, addressing everything from system invasions, data theft, to financial crimes and ransomware attacks. These crimes, often committed remotely and anonymously, make it difficult to identify those responsible and require advanced digital investigation techniques. The study highlights the importance of integrating technical and legal knowledge so that investigations are effective and comply with legal principles, ensuring the validity of digital evidence collected in the fight against cybercrimes. In this way, the work contributes to the understanding of practices and limitations in the current scenario of forensic computing applied to encrypted media, providing theoretical support for action in real cases of cybercrimes.

Keywords: computer forensics; cybercrimes; fuzzy hash; data manipulation; digital evidence.

LISTA DE FIGURAS

TABELAS

SUMÁRIO

1 INTRODUÇÃO

A popularização da internet, ocorrida nos anos 1990 com a criação do serviço World Wide Web (WWW) por Lee (1989), permitiu que usuários em diferentes partes do mundo pudessem trocar dados e informações em milissegundos, promovendo maior velocidade na comunicação entre máquinas e, consequentemente, entre as pessoas (Eleutério; Machado, 2011).

> No Brasil, assim como em diversas partes do mundo, vivencia-se a era digital, que proporcionou inúmeros benefícios à sociedade. No entanto, o amplo acesso à tecnologia também contribuiu para o surgimento de diferentes modalidades de crimes cibernéticos, conforme apontam Barreto e Santos (2019).

Isso evidenciou a necessidade de uma área especializada capaz de investigar tais delitos e formar profissionais aptos a realizar análises periciais em ambientes digitais.

> A forense computacional em imagens refere-se ao processo de investigação e análise de arquivos digitais, com o objetivo de coletar evidências que possam ser utilizadas em procedimentos judiciais. Essa área abrange atividades como a recuperação de dados ocultos, autenticação de imagens, extração de metadados, detecção de conteúdos ilícitos e identificação de manipulações (Mayer; Stamm, 2020).

Com o avanço da tecnologia, o acesso à internet tornou-se amplamente difundido entre pessoas de todas as idades, por meio de dispositivos como smartphones, notebooks e tablets. A digitalização das atividades cotidianas, como comunicação, pagamentos, pesquisa e ensino, trouxe praticidade e eficiência. Contudo, esse crescimento também favoreceu o aumento dos crimes digitais, frequentemente praticados por indivíduos mal-intencionados que utilizam a rede para aplicar golpes, cometer crimes de aliciamento, pedofilia, entre outros. Nesses casos, a perícia forense digital desempenha papel fundamental na identificação e responsabilização dos envolvidos.

A análise forense digital compreende a coleta, preservação, exame e apresentação de evidências digitais, as quais podem incluir arquivos, registros de

atividades online, e metadados. Essa análise é essencial para assegurar a integridade das provas e auxiliar na identificação dos autores de crimes cibernéticos.

Dentre as diversas abordagens existentes, destacam-se os algoritmos de hash de similaridade, que possibilitam a identificação de arquivos com conteúdo parcialmente semelhante. Essa técnica é dividida em algumas implementações: ssdeep, sdhash e TLSH, cada uma com abordagens distintas para comparar e agrupar dados digitais.

Uma função hash é um algoritmo que transforma uma entrada (ou "mensagem") de tamanho variável em uma saída de tamanho fixo, normalmente representada como uma sequência de caracteres. Essa saída é chamada de "valor hash" ou "digest" (Rivest, 1992). As funções hash são amplamente utilizadas em áreas como segurança de dados, integridade de arquivos, indexação em bases de dados e criptografia (Kaufman, Perlman & Speciner, 2014).

> As principais características de uma função hash é que a produção de uma saída deve ser rápida, deve gerar valores de saída uniformemente distribuídos, e, preferencialmente, deve ser "irreversível" (ou seja, não deve ser possível reconstruir a entrada original a partir do valor hash) (Menezes, van Oorschot & Vanstone, 1997)

> As funções hash são fundamentais para a proteção da integridade e autenticidade de informações no ambiente digital. Elas consistem em algoritmos que transformam entradas de comprimento variável em saídas de tamanho fixo, atuando como uma "impressão digital" dos dados originais. Essa transformação deve ser rápida, eficiente e garantir que pequenas alterações na entrada resultem em saídas completamente diferentes, de modo a evitar fraudes ou manipulações de dados (Menezes; Van oorschot; Vansyone, 1996).

Essas funções devem atender a propriedades de segurança específicas, como a resistência à colisão, resistência à pré-imagem e resistência à segunda pré-imagem. A resistência à colisão, por exemplo, implica que deve ser extremamente difícil encontrar duas entradas distintas que resultem no mesmo hash.

“Já a resistência à pré-imagem dificulta a obtenção da entrada original a partir do valor do hash gerado, aumentando a confiabilidade do método na proteção de dados sensíveis. ” (Paar; Pelzl, 2010).

A segurança das funções hash é um elemento vital no funcionamento de protocolos de comunicação segura e sistemas criptográficos. Sem essas propriedades, seria possível que dados fossem alterados sem detecção, comprometendo a confiança em transações digitais, sistemas bancários, autenticação de usuários, e diversas outras aplicações (Knudsen; Robshaw, 2011).

No campo da perícia forense digital, as funções hash são utilizadas para assegurar a autenticidade das evidências coletadas em investigações. Ao gerar o hash de um arquivo ou disco rígido, é possível demonstrar posteriormente que o material analisado permaneceu íntegro durante todo o processo pericial, o que é crucial para a validade jurídica da prova digital (Casey, 2011).

Entre as muitas abordagens empregadas na perícia digital, os algoritmos de *hash de similaridade* se destacam, pois permitem identificar arquivos com conteúdos semelhantes, mesmo após pequenas mudanças. Diferente das funções hash comuns, que geram resultados diferentes mesmo que os dados comparados sejam bastante similares, os algoritmos de similaridade foram criados para ver relações aproximadas entre arquivos, o que é bom para analisar documentos digitais que foram alterados ou escondidos (Roussev, 2010).

Essa técnica vem ganhando destaque com o aumento dos crimes virtuais, onde conteúdos ilegais são frequentemente editados, têm seus nomes alterados ou são disfarçados com o intuito de dificultar sua identificação por mecanismos convencionais de verificação. Os hashes de similaridade são ferramentas que, além de garantir a integridade dos dados, deixam comparar arquivos, mesmo que não sejam iguais, ajudando a achar padrões de reutilização ou modificação do conteúdo (Breitinger; Baier, 2013).

O algoritmo SSDEEP é um dos mais reconhecidos no campo dos hashes de similaridade. Baseado na técnica de Context Triggered Piecewise Hashing (CTPH), ele divide os arquivos em blocos menores e calcula valores hash individualmente. Este método permite detectar similaridades mesmo quando ocorrem pequenas modificações no conteúdo do arquivo, como a substituição de palavras ou inserção de dados. Por isso, é abundante utilizado na comparação de variantes de malware, documentos alterados e outros arquivos na qual a estrutura possa ter sido alterada com o intuito de mascarar o conteúdo original (Roussev, 2010).

Apesar de sua popularidade e eficiência, o ssdeep apresenta algumas limitações, especialmente em relação à precisão, podendo gerar falsos positivos. Por esse motivo, pesquisadores têm buscado formas de aprimorar o algoritmo, ajustando sua sensibilidade e critérios de segmentação para reduzir erros e aumentar a confiabilidade dos resultados (Breitinger; Baier, 2013).

Já o ***SDHASH*** (Similarity Digest Hashing), proposto por Roussev (2010), é uma das ferramentas de Approximate Matching (AM) mais consolidadas na área de perícia forense digital. Seu objetivo é representar objetos digitais por meio de digests formados a partir de sequências de bytes com alta entropia — menos suscetíveis a ocorrerem aleatoriamente — o que possibilita detectar similaridades mesmo em arquivos modificados parcialmente.

Nesse contexto, surge o ***TLSH*** *(Trend Locality Sensitive Hashing)*, proposto por Oliver et al. (2013), como uma técnica eficiente baseada no conceito de locality-sensitive hashing (LSH). O TLSH gera um digest de tamanho fixo (35 bytes) que representa características de um objeto de forma que objetos semelhantes apresentem digests também semelhantes, mesmo diante de pequenas modificações.

Na investigação criminal moderna, compreender o papel da tecnologia é essencial para garantir a efetividade das ações periciais. As ferramentas e metodologias utilizadas atualmente são aliadas importantes na produção de provas técnicas, contribuindo diretamente para o esclarecimento de fatos e a promoção da justiça. Por meio de métodos avançados, como a análise de dados digitais, identificação biométrica e o uso de algoritmos de hash de similaridade, é possível obter evidências mais confiáveis e precisas no contexto criminal.

O intuito deste estudo é desvendar a forense computacional, com foco nos algoritmos de hash de similaridade SSDEEP, SDHASH e TLSH, mais precisamente na sua parte teórica, técnicas, vantagens, e, claro, suas limitações. A pesquisa foca em como essas ferramentas ajudam a identificar arquivos parecidos, mesmo após modificações realizadas com o objetivo de esconder conteúdo potencialmente malicioso.

Além da análise dos algoritmos, essas ferramentas serão comparadas com o objetivo de avaliar suas características, níveis de acerto e desempenho. A ideia é dar uma avaliação crítica, que possa auxiliar na melhor abordagem em cada investigação, impulsionando as práticas forenses e incentivando novos estudos.

Na seção 2, serão abordados os objetivos gerais bem como os específicos. A Seção 3 apresenta a fundamentação teórica, abordando os principais conceitos relacionados à perícia forense digital e aos algoritmos de hash de similaridade. Em seguida, a Seção 4 detalha a metodologia utilizada no desenvolvimento deste trabalho, com o intuito de garantir a coerência e a

validade dos resultados obtidos. A Seção 5 apresenta os resultados da prova de conceito, com análise aprofundada do comportamento de cada algoritmo. Por fim, a Seção 6 discute as implicações dos achados, contribuições do trabalho, limitações identificadas e propostas para trabalhos futuros.

2 OBJETIVOS

2.1 Objetivo Geral

O trabalho tem por objetivo analisar a aplicação de algoritmos de hash de similaridade, com foco em SSDEEP, SDHASH e TLSH, no contexto da forense computacional, a fim de compreender suas contribuições, limitações e efetividade na identificação de arquivos com conteúdo parcialmente semelhante em investigações digitais.

2.2 Objetivos Específicos:

- Apresentar conceitos da análise forense digital e o papel dos algoritmos de hash neste contexto;
- Descrever os funcionamentos dos algoritmos ssdeep, sdhash e TLSH;
- Comparar os algoritmos ssdeep, sdhash e TLSH quanto à precisão, desempenho e aplicabilidade em diferentes cenários forenses.
- Identificar as vantagens e limitações de cada algoritmo na detecção de arquivos alterados ou semelhantes.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 Forense Computacional

A análise forense digital é o processo de coleta, preservação, análise e apresentação de evidências digitais que podem ser utilizadas em investigações criminais ou processos judiciais. Essas evidências podem incluir arquivos de computador, registros de atividades online, metadados, entre outros. A análise forense digital é essencial para identificar a autoria de crimes cibernéticos e garantir a integridade das provas coletadas.

São utilizadas múltiplas técnicas na análise forense digital para extrair e analisar evidências digitais de forma eficiente. Algumas das principais técnicas incluem a recuperação de arquivos deletados, a análise de metadados, a identificação de padrões de comportamento, entre outras. Essas técnicas permitem aos profissionais identificar possíveis suspeitos, reconstruir eventos e apresentar provas sólidas em processos judiciais.

A análise forense digital segue um processo estruturado para garantir a integridade das evidências, como:

- **Preservação de evidências:** O primeiro passo é a preservação dos dados em seu estado original. Isso pode envolver o isolamento do dispositivo, a criação de imagens forenses (cópias exatas) dos discos rígidos, e a documentação detalhada do processo.
- **Aquisição de dados:** Depois da preservação, copia-se os dados para um ambiente seguro para análise sem risco de alteração. Ferramentas especializadas são usadas para garantir que a cópia seja fiel ao original.
- **Análise de dados:** Nesta etapa, examina-se os dados em busca de evidências relevantes, utilizando técnicas como a recuperação de arquivos deletados, análise de logs de sistema, e decodificação de dados criptografados.
- **Documentação e relatório:** Todos os passos do processo, desde a coleta até a análise, são documentados detalhadamente. O relatório final deve ser claro e compreensível, apresentando as descobertas para investigações ou processos judiciais.

3.2 Hash

> Uma função hash é um algoritmo que mapeia os dados de comprimento variável para dados de comprimento fixo. Os valores retornados por uma função hash são chamados valores hash, códigos hash, somas hash, ou simplesmente hashes (Rsa Laboratories, Bulletin 4, 1996).

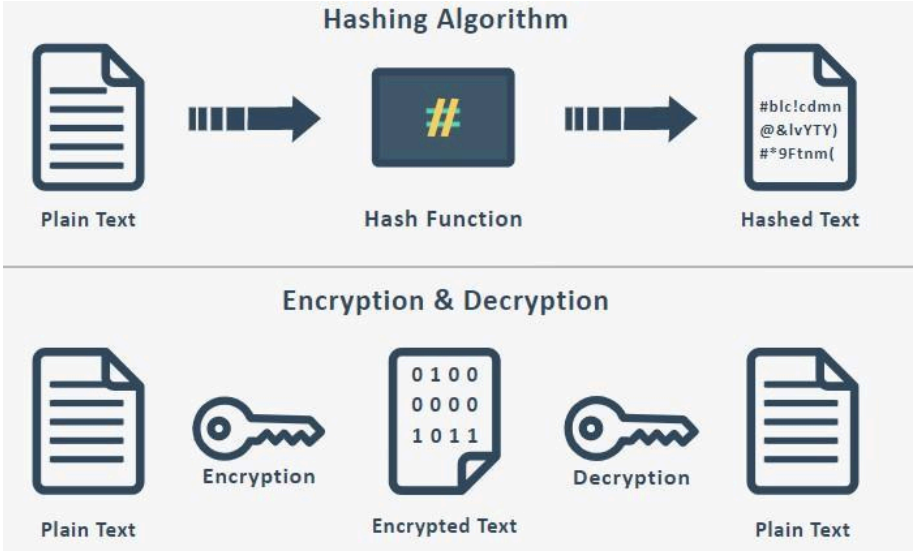
Tecnicamente, uma hash seria a transformação de um grande volume de dados para um pequeno volume de informação. Para manter a integridade de um hash, que é uma sequência de bits, eles são gerados por um algoritmo de dispersão. De modo geral ele é representado na base hexadecimal. Tal sequência tem como objetivo identificar um arquivo ou uma informação de forma quase exclusiva (EMC², “O que é uma função hash?”, 2013).

> As funções hashes são muito utilizadas para acelerar o processo de assinaturas digitais em documentos, mas também podem ser utilizadas em uma vasta variedade de assuntos, tais como: em senhas, correio eletrônico, criação de chaves criptográficas, transferência de arquivos, etc (Rsa Laboratories, Bulletin 4, 1996)

Além de servir como um mecanismo de segurança robusto, esse método de criptografia garante a integridade do documento e assegura que seu conteúdo permaneça inalterado desde o momento da assinatura. Isso é essencial em contextos empresariais, em que a validade e a autenticidade dos documentos precisam ser mantidas.

O processo de hashing, que envolve a transformação dos dados de entrada em uma 'impressão digital' única, é essencial para garantir a integridade da informação, conforme ilustrado na **Figura 1**.

Figura 1 – Exemplo ilustrativo do funcionamento de uma função hash



Fonte: Akad, 2023

A Figura 1 ilustra como a transformação de dados em um hash gera uma "impressão digital" única e de tamanho fixo. Este processo é ideal para verificar a integridade exata de um arquivo. No entanto, em uma investigação forense, onde dados podem ser levemente alterados, essa abordagem se torna limitada. Por isso, a próxima seção aborda o conceito de hashing de similaridade, que busca comparar não apenas arquivos idênticos, mas aqueles que porventura possam ter passado por pequenas alterações.

3.3 Hash de similaridade

O hash de similaridade, ou *fuzzy hashing*, é uma técnica de resumo digital que permite comparar arquivos com base em sua semelhança de conteúdo, mesmo que não sejam idênticos em nível binário. Diferentemente das funções de hash tradicionais, como MD5 ou SHA-1, que produzem saídas completamente diferentes a partir de pequenas modificações (devido ao efeito avalanche), os algoritmos de hash de similaridade geram resumos que preservam certo grau de correlação com o conteúdo original (Roussev, 2010; Kornblum, 2006; Tridgell et al., 2013).

As funções de hash convencionais são eficientes para garantir integridade, mas não são adequadas para detectar similaridades entre arquivos parcialmente modificados. Por isso, em contextos como análise forense digital, detecção de variantes de malware e deduplicação de dados, os hashes de similaridade se tornam mais apropriados (Kornblum, 2006).

O algoritmo ssdeep, proposto por Kornblum (2006), foi um dos primeiros a utilizar conteúdo dividido em blocos dinâmicos e aplicação de hash sobre esses blocos. O processo utiliza uma janela deslizante para determinar pontos de divisão baseados em conteúdo, gerando assinaturas que podem ser comparadas por distância de edição. Apesar de eficaz em muitos cenários, o ssdeep possui limitações quanto à granularidade de análise e à resistência a modificações maiores.

Para superar essas limitações, Roussev (2010) propôs o algoritmo **sdhash**, que seleciona *features* de baixa entropia – sequências de bytes mais informativas – e as insere em filtros de Bloom. A similaridade entre dois arquivos é então determinada pela sobreposição dos filtros, retornando uma pontuação proporcional à semelhança. A noção de entropia, proposta por Shannon (1948), refere-se à quantidade média de informação ou incerteza presente em uma sequência de símbolos. Dados altamente previsíveis, como longas sequências de zeros, apresentam baixa entropia, enquanto dados mais aleatórios, como arquivos criptografados ou comprimidos, apresentam alta entropia.

> No contexto do fuzzy hashing, trechos de baixa entropia são preferidos porque contêm padrões representativos e discriminativos, enquanto regiões de alta entropia pouco contribuem para a diferenciação, por serem semelhantes entre si (Shannon, 1948; Roussev, 2010; Broder & Mitzenmacher, 2004).

Mais recentemente, foi desenvolvido o algoritmo **TLSH** (Trend Locality Sensitive Hashing), com foco em escalabilidade e aplicação em grandes volumes de dados. TLSH utiliza a análise de distribuição de bytes e geração de um digest que leva em conta frequência, ordem local e normalização.

“Ele é especialmente eficaz em contextos como deduplicação, agrupamento e detecção de variantes, superando limitações anteriores relacionadas à sensibilidade a pequenas alterações” (Tridgell et al., 2013).

“Essas técnicas se destacam por possibilitar a detecção de conteúdo similar, mesmo quando o objeto é parcialmente truncado, corrompido ou sofreu modificações simples como inserções e exclusões de trechos” (Roussev, 2010).

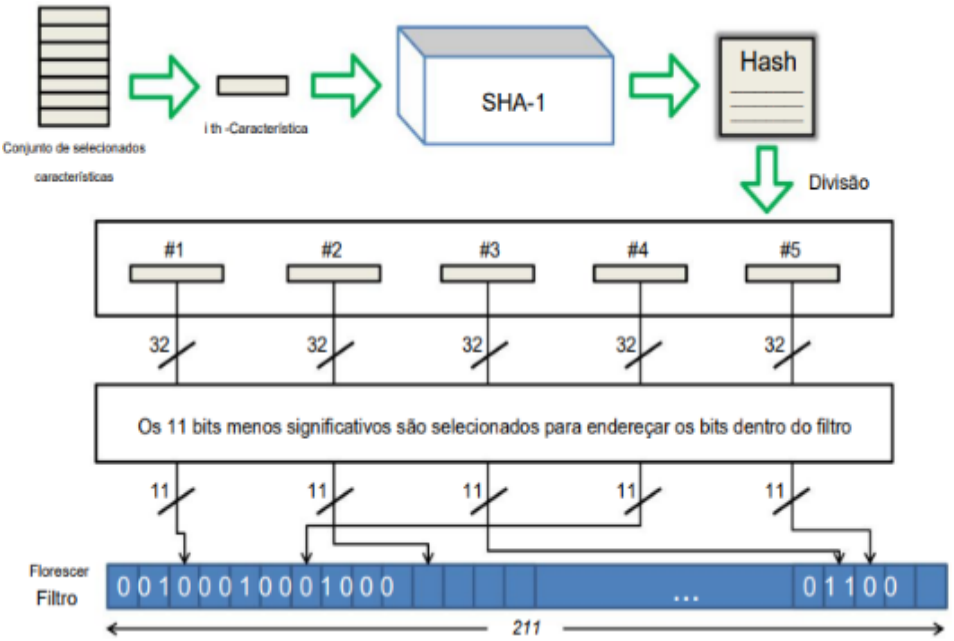
3.3.1 SDHASH

O sdhash (Similarity Digest Hash) é uma ferramenta projetada para comparar objetos digitais com base em similaridade de conteúdo, e não igualdade exata. Desenvolvido por Vassil Roussev em 2010, o sdhash seleciona e codifica *features* (sequências de bytes) que são raras ou distintivas dentro de um objeto digital, usando como critério a entropia de Shannon.

Termos principais:

- Feature: sequência de bytes extraída do objeto (tamanho padrão: 64 bytes).
- W (window size): tamanho da janela deslizante usada para extrair *features* com menor entropia (padrão: 64).
- m: tamanho do filtro de Bloom (geralmente em bits).
- k: número de funções de hash usadas para marcar bits no filtro de Bloom.
- fmax: número máximo de *features* por filtro de Bloom.
- t: limiar mínimo (threshold) de popularidade para uma *feature* ser considerada (padrão: 16).
- Rprec (rank de precedência): ordena as *features* com base no valor de entropia.
- Rpop (rank de popularidade): número de vezes que a *feature* teve a menor entropia no contexto da janela.

Figura 2 – Exemplo ilustrativo do funcionamento de sdhash



Fonte: Moia 2023)

Processo de geração do *digest*:

1. **Extração de features:**

    - Uma janela de tamanho fixo percorre o objeto byte a byte.

    - Cada segmento extraído recebe um valor de entropia calculado por Shannon (normalizado entre 0 e 1000).

2. **Filtragem de features:**

    - *Features* com entropia muito baixa (≤100) ou muito alta (>990) são descartadas, pois são consideradas pouco úteis ou muito comuns.

3. **Seleção de features:**
    - A cada posição da janela deslizante, a *feature* com menor entropia tem seu contador Rpop incrementado.

    - Apenas as *features* com Rpop ≥ t são selecionadas para o digest.

3.1. Filtro de Bloom

- Um **filtro de Bloom** é uma **estrutura de dados probabilística** usada para verificar rapidamente se um elemento possivelmente pertence a um conjunto.

- Ele é composto por um vetor de bits inicialmente zerados e diversas funções de hash.

- Cada elemento inserido gera múltiplos bits marcados como “1”.

- Essa técnica permite verificar de forma eficiente se duas coleções compartilham elementos semelhantes.

- No **sdhash**, o filtro de Bloom armazena as features selecionadas, possibilitando comparar objetos digitais pela sobreposição dos bits — quanto maior a interseção, maior a similaridade.

4. **Codificação das features:**

    - Cada *feature* selecionada é processada por um hash (SHA-1).

- Os bits resultantes são divididos e usados para setar posições específicas em um filtro de Bloom.

- Quando o filtro atinge o limite de *features* (fmax), um novo filtro é criado.

- O digest final é a concatenação de todos os filtros.

3.3.2 SSDEEP

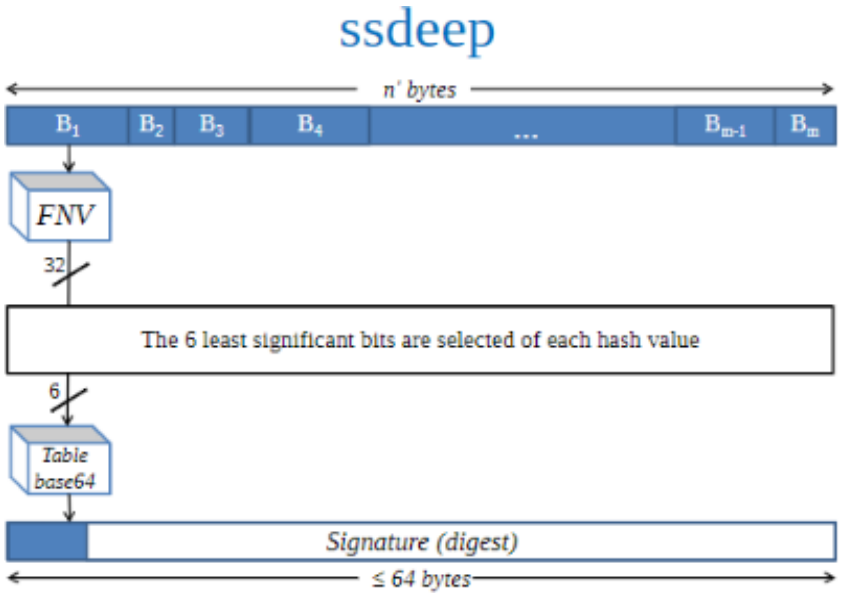
"O algoritmo ssdeep foi desenvolvido por Jesse Kornblum em 2006, com base em um método originalmente proposto por Andrew Tridgell para detecção de spam por similaridade de conteúdo" (Tridgell, 2002; Kornblum, 2006). Essa ferramenta implementa o método conhecido como Content-Triggered Piecewise Hashing (CTPH), uma técnica de fragmentação e hash que busca detectar similaridades entre arquivos a nível de byte, mesmo na presença de pequenas alterações.

A principal característica do ssdeep é sua capacidade de gerar blocos de tamanhos variáveis a partir de uma função de rolling hash — ou hash deslizante — que determina os pontos de início e fim de cada bloco. Essa função calcula valores de hash com base em uma janela deslizante de tamanho fixo (7 bytes), que percorre o arquivo byte a byte. "Quando essa função produz um valor específico (conhecido como trigger), um novo bloco é delimitado. Cada bloco, então, é processado com a função de hash FNV" (Noll, 2012), e os 6 bits menos significativos do resultado são codificados em caracteres Base64. A digest final é composta pela concatenação dos caracteres gerados a partir de todos os blocos identificados.

Para comparar dois digests, o ssdeep emprega o algoritmo de Distância de Edição (Edit Distance) proposto por Ukkonen (1983). Essa métrica calcula o número mínimo de operações necessárias para transformar uma string em outra, considerando inserções, deleções, substituições e transposições de caracteres. O resultado da comparação é uma pontuação de similaridade que varia de 0 (totalmente diferentes) a 100 (idênticos).

Uma das vantagens do ssdeep em relação aos métodos tradicionais baseados em blocos fixos é sua menor sensibilidade a problemas de alinhamento e sua resistência a pequenas inserções ou deleções. No entanto, sua eficácia está limitada a objetos de tamanho relativamente pequeno e similar, devido à dependência de parâmetros derivados do tamanho do arquivo. Para contornar essa limitação, o ssdeep gera dois digests por objeto, utilizando dois valores distintos como pontos de gatilho: um correspondente ao tamanho do bloco base (b) e outro ao dobro desse valor (2b). Assim, o algoritmo consegue comparar objetos cujos blocos diferem em até um fator de dois. A Figura 3 ilustra esse processo, representando como o arquivo é dividido em blocos variáveis pela função de *rolling hash*, processados pela função FNV e convertidos em caracteres Base64 até a formação do digest final.

Figura 3 – Exemplo ilustrativo do funcionamento de ssdeep



Fonte: Moia (2023)

“Apesar de sua ampla utilização na área de forense digital e detecção de similaridade, o ssdeep apresenta limitações que têm sido objeto de estudo. Pesquisas apontam problemas de desempenho” (Chen; Wang, 2008; Breitinger; Baier, 2012) e vulnerabilidades a ataques ativos, como demonstrado por Baier e Breitinger (2011).

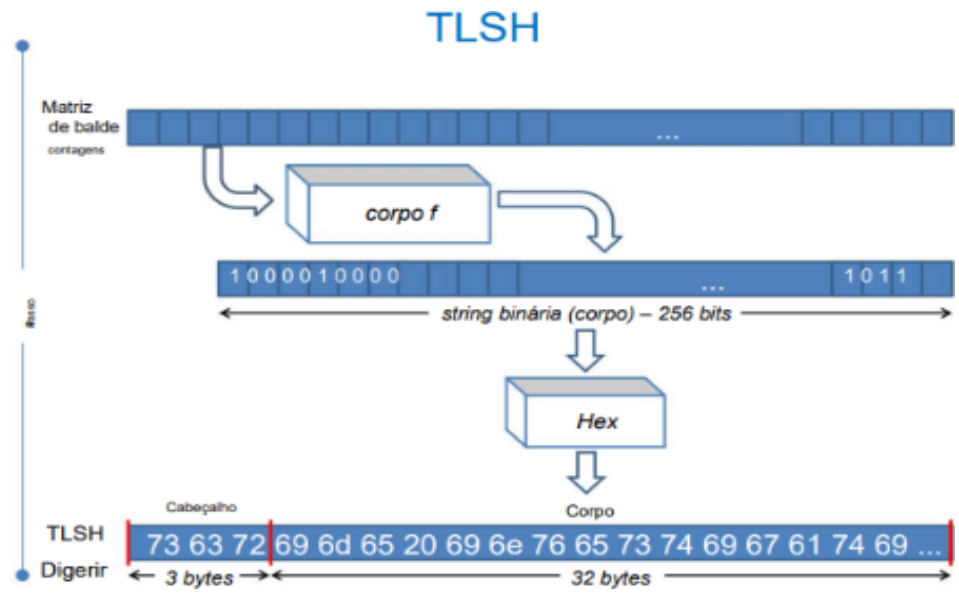
3.3.3 TLSH

Proposto por Oliver et al. (Oliver et al., 2013), o TLSH (Trend Locality Sensitive Hash) é um algoritmo baseado no conceito de Locality Sensitive Hashing (LSH), cujo objetivo principal é gerar um resumo (digest) de tamanho fixo para identificar similaridades entre objetos digitais. Diferentemente de técnicas tradicionais de hashing que visam detectar igualdade exata, o TLSH é projetado para capturar semelhanças entre arquivos ou dados que sofreram alterações parciais.

O processo de criação do digest inicia com uma janela deslizante de 5 bytes, que percorre o objeto de entrada byte a byte. Em cada passo, são extraídos seis trigramas — combinações de três caracteres consecutivos — a partir da janela. Esses trigramas são então mapeados para um array de contadores de 128 posições (buckets) utilizando a função de hash de Pearson (Pearson, 1990). Em seguida, são calculados três pontos de quartis (q1, q2 e q3), com base na distribuição dos valores no array de contadores. Essa etapa é fundamental para caracterizar a "assinatura" estatística do objeto, conforme ilustrado por Oliver et al. (2013).

O digest final gerado pelo TLSH possui 35 bytes, divididos em duas partes: cabeçalho (header) e corpo (body). O cabeçalho contém informações resumidas como os três quartis, o tamanho do objeto e um valor de verificação (checksum), totalizando 3 bytes. Já o corpo, com 32 bytes, é construído comparando-se os valores dos contadores aos quartis e codificando os resultados em pares de bits, representando a distribuição dos dados em relação aos quartis definidos. A Figura 4 ilustra esse processo, destacando as etapas de extração de trigramas, cálculo dos quartis e construção do digest final.

Figura 4: Exemplo ilustrativo do funcionamento de TLSH



Fonte: Vitor Hugo Galhardo Moia 2023)

Para comparação entre dois objetos, o TLSH utiliza duas funções principais: a Distance Header, que considera as diferenças nos quartis e no tamanho dos objetos, e a Distance Body, que estima uma distância de Hamming entre os corpos dos digests. A soma dessas duas distâncias fornece a métrica final de similaridade. Um valor próximo de 0 indica objetos idênticos ou muito semelhantes, enquanto valores maiores representam maior dissimilaridade.

Pesquisas subsequentes (OLIVER et al., 2014) demonstraram que o TLSH é mais robusto a alterações aleatórias e manipulações adversariais do que outros algoritmos como o ssdeep e o sdhash. No entanto, uma limitação importante do TLSH é sua baixa eficácia na detecção de similaridade por contenção, ou seja, quando um objeto está parcialmente contido em outro. Estudos como o de Lee e Atkison (Lee; Atkison, 2017) reforçam essa observação, evidenciando uma capacidade limitada do TLSH nesse aspecto específico.

4 METODOLOGIA

Diante dos conceitos explorados na fundamentação teórica, torna-se necessário verificar, na prática, o comportamento dessas ferramentas no contexto da análise forense digital. Para isso, este trabalho propõe a realização de uma prova de conceito que permita observar a capacidade de cada algoritmo em identificar modificações em arquivos digitais.

A seguir, descreve-se a abordagem metodológica adotada nesta pesquisa, contemplando os aspectos teóricos e práticos, os procedimentos que serão aplicados e as ferramentas previstas para a etapa experimental.

4.1 Tipo de Pesquisa

A pesquisa é classificada como aplicada, pois visa gerar conhecimento voltado à prática da perícia digital. A etapa exploratória contempla a revisão teórica dos algoritmos SSDEEP, SDHASH e TLSH, enquanto a etapa experimental será dedicada à validação prática dos conceitos estudados.

4.2 Abordagem Metodológica

- **Fase teórica**: Foi realizado um levantamento de informações técnicas e científicas sobre os algoritmos de fuzzy hashing, com ênfase em suas aplicações na detecção de manipulações em arquivos digitais. Essa etapa permitiu compreender os princípios de funcionamento dos algoritmos, suas limitações e vantagens, fornecendo a base necessária para a implementação e avaliação experimental na fase prática.

- **Fase prática**: Foi desenvolvido um protótipo em Python destinado a avaliar a sensibilidade e a robustez dos algoritmos ssdeep, sdhash e tlsh frente a diferentes tipos e intensidades de alterações em arquivos digitais. Implementou-se um script para alteração contínua de bytes (permite especificar a porcentagem e o padrão de alteração) e gerou-se variantes dos arquivos originais. O código

para geração das variantes e para execução dos testes do tlsh e sdhash foi desenvolvido e executado no ambiente online Replit, enquanto os testes com ssdeep foram realizados no ambiente local utilizando o Visual Studio Code. Para cada variante foram calculados e registrados os scores de similaridade, organizados em planilhas com informações sobre o cenário, intensidade da alteração e resultados por algoritmo. Os dados obtidos foram analisados quantitativamente (médias, desvio padrão, distribuições) e qualitativamente, permitindo comparar a performance dos algoritmos e identificar limites práticos de detecção.

4.3 Procedimentos Aplicados na Prova de Conceito

A prova de conceito foi implementada com base nos seguintes passos:

- Criação de arquivos originais e suas versões modificadas por meio de alterações contínuas de bytes, simulando diferentes níveis de modificação.
- Aplicação dos algoritmos ssdeep, sdhash e tlsh sobre os arquivos para geração dos respectivos digests. Os testes com tlsh e sdhash foram realizados no ambiente Replit, enquanto os testes com ssdeep foram realizados no Visual Studio Code.
- Comparação de similaridade entre os arquivos, utilizando os scores retornados por cada algoritmo.

4.4 Ferramentas e Tecnologias

- Linguagem: Python
- Ferramentas: ssdeep, sdhash e tlsh
- Sistema operacional: Replit e Visual Studio Code

5 RESULTADOS DA PROVA DE CONCEITO

A prova de conceito permitiu avaliar o comportamento dos algoritmos de fuzzy hashing SSDEEP, SDHASH e TLSH quando aplicados a diferentes formatos de arquivos digitais (CSV, imagens, PDF e áudio), submetidos a alterações graduais de 5%, 10%, 20%, 40% e 50%. A análise dos resultados revela não apenas diferenças quantitativas nos scores de similaridade, mas também padrões de comportamento que refletem diretamente os princípios de funcionamento de cada algoritmo e as características estruturais dos formatos de arquivo analisados.

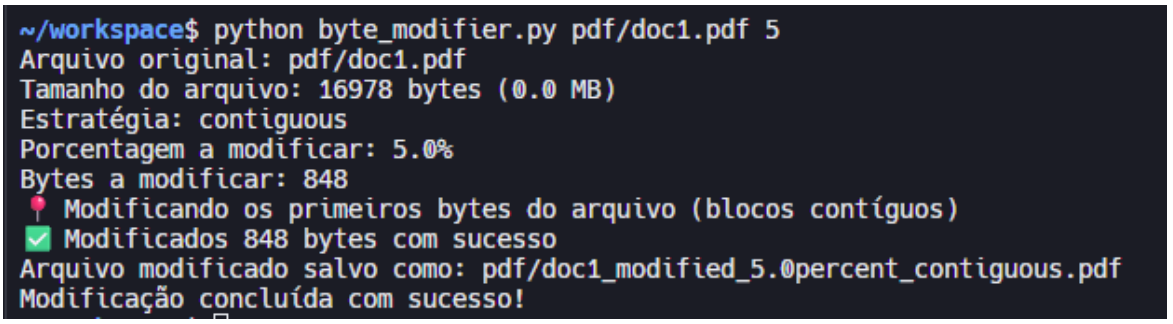
5.1 Alteração de Bytes

Nesta etapa da prova de conceito, foi desenvolvido e executado um script em Python (.py) responsável por realizar alterações contínuas de bytes em arquivos digitais de diferentes formatos — CSV, imagens, PDF e áudio. O objetivo foi avaliar o comportamento dos algoritmos de *fuzzy hashing* SSDEEP, SDHASH e TLSH diante de modificações controladas no conteúdo binário dos arquivos.

O script foi programado para substituir segmentos consecutivos de bytes do arquivo original por novos valores, de forma incremental e reprodutível, gerando versões alteradas com níveis de modificação de 5%, 10%, 20%, 40% e 50%. Essa abordagem simulou cenários de adulteração gradual, mantendo a estrutura geral dos arquivos intacta, mas modificando significativamente sua composição binária interna.

Após a geração das versões modificadas, cada uma delas foi comparada com o arquivo original utilizando os três algoritmos de *fuzzy hashing*, permitindo calcular os scores de similaridade e observar o impacto das alterações progressivas sobre a capacidade de detecção de correspondência.

Figura 5:Resultado da alteração de bytes contínuos



Fonte: Elaborado pelo autor (2025)

Essa metodologia possibilitou avaliar de forma sistemática a sensibilidade e a tolerância dos algoritmos a alterações contínuas de bytes, fornecendo uma base experimental sólida para interpretar seus limites de detecção em diferentes tipos de arquivo.

Os resultados obtidos são apresentados nas subseções seguintes.

5.2 Resultados SSDEEP

O SSDEEP apresentou desempenho altamente dependente do tipo de arquivo e da natureza das modificações aplicadas. Para alterações de baixa intensidade (5% e 10%), o algoritmo manteve scores superiores a 90% em arquivos de imagem e áudio, demonstrando eficácia na detecção de similaridade quando a estrutura geral do arquivo permanece preservada.

No entanto, a queda acentuada observada nos arquivos CSV testados, especialmente a partir de 20% de modificação, pode ser explicada pelo mecanismo de funcionamento do SSDEEP baseado em context-triggered piecewise hashing (CTPH). Este algoritmo divide o arquivo em blocos variáveis utilizando rolling hash com base em pontos de "gatilho" no conteúdo. Quando alterações em dados tabulares ocorrem, há alta probabilidade de que esses pontos de divisão sejam deslocados, resultando em uma reorganização completa dos blocos e, consequentemente, em hashes radicalmente diferentes para cada segmento.

Arquivos de imagem e áudio, por outro lado, apresentam maior redundância de padrões binários e estruturas mais homogêneas. Alterações pontuais nesses formatos tendem a afetar blocos específicos sem necessariamente deslocar todos os pontos de divisão, preservando assim a similaridade de outros segmentos. Isso explica por que o SSDEEP mantém scores mais elevados nesses formatos mesmo com modificações de até 20%.

A queda para valores nulos em certos arquivos CSV com 40-50% de alteração indica que o SSDEEP atingiu seu limite de detecção, onde a mudança estrutural foi tão significativa que os blocos resultantes não compartilham características suficientes para estabelecer correlação. Este comportamento é consistente com a literatura técnica do algoritmo, que reconhece limitações em cenários de alterações distribuídas que causam realinhamento dos blocos de hash.

Com relação à análise forense de documentos, o SSDEEP é adequado para detecção de modificações sutis em arquivos multimídia, mas apresenta limitações significativas para análise de arquivos estruturados como CSV com alterações moderadas a extensas. Em contextos periciais onde se suspeita de manipulação substancial de dados tabulares, o uso isolado do SSDEEP pode resultar em falsos negativos.

Parâmetros Utilizados no SSDEEP

A implementação do SSDEEP analisada utiliza parâmetros específicos de linha de comando para geração e comparação de hashes fuzzy através do binário ssdeep.exe.

Parâmetros de Linha de Comando

Parâmetro -s (Silent Mode): Ativa o modo silencioso, suprimindo mensagens de status e avisos durante a execução. É utilizado tanto na geração de hashes quanto na comparação de arquivos.

Parâmetro -c (CSV Output): Configura a saída no formato CSV, facilitando o parsing dos resultados. Permite extrair facilmente os componentes do hash gerado, incluindo o blocksize e os valores de hash.

Parâmetro -x (Compare Mode): Ativa o modo de comparação entre arquivos ou hashes, permitindo calcular a similaridade percentual entre dois conjuntos de dados.

Estrutura de Dados

A constante SSDEEP_HEADERS define o formato de cabeçalho: "ssdeep,1.1--blocksize:hash:hash,filename", especificando a versão do SSDEEP (1.1) e o formato dos dados.

Parâmetros Implícitos do Algoritmo

Blocksize: Parâmetro fundamental que determina o tamanho dos blocos utilizados no processamento. É calculado automaticamente pelo algoritmo baseado no tamanho do arquivo de entrada. Aparece como o primeiro componente do hash gerado no formato "blocksize:hash1:hash2".

Hashes Duplos: O SSDEEP gera dois hashes distintos para cada arquivo, permitindo comparações em diferentes escalas e aumentando a sensibilidade para detectar similaridades parciais.

Métrica de Similaridade: Expressa como valor percentual entre 0 e 100, onde 0 indica arquivos completamente diferentes e 100 representa arquivos idênticos. Valores acima de 50 geralmente indicam correspondência significativa.

Tabela 1: Resultados dos testes com SSDEEP

CSV						
	Nome do arquivo	**5%**	**10%**	**20%**	**40%**	**50%**
1	test1	91	91	74	40	35
2	test2	96	90	80	65	58
3	test3	0	0	0	0	0
4	test4	71	52	0	0	0
5	test5	91	79	66	0	40
7	test6	79	61	0	0	0
8	test7	91	91	77	63	54
9	test8	0	0	0	0	0
10	test9	65	0	0	0	0

Imagem - JPG						
	Nome do arquivo	**5%**	**10%**	**20%**	**40%**	**50%**
1	img	97,00%	94	83	74	72
2	img2	99	86	75	60	55
3	img3	96	91	82	66	54
4	img4	94	97	83	66	66
5	img5	93	90	75	69	50
6	img6	96	96	82	66	57
7	img7	93	96	88	69	68
8	img8	97	90	80	66	58
9	img9	96	93	82	61	57
10	img10	96	88	83	55	63

PDF						
	Nome do arquivo	**5%**	**10%**	**20%**	**40%**	**50%**
1	doc1	91	88	85	54	60
2	doc2	96	85	86	72	65
3	doc3	93	88	77	66	54
4	doc4	93	91	85	66	60
5	doc5	97	88	85	66	58
6	doc6	93	90	69	47	36
7	doc7	93	91	74	68	50
8	doc8	96	91	80	61	58
9	doc9	94	93	88	69	60
10	doc10	91	90	74	63	60

Audio						
	Nome do arquivo	**5%**	**10%**	**20%**	**40%**	**50%**
1	audio1	96	86	82	75	58
2	audio2	96	94	85	66	55
3	audio3	90	90	79	60	58
4	audio4	96	91	88	68	71
5	audio5	96	96	85	65	57
6	audio6	91	88	83	65	63
7	audio7	96	94	85	69	69
8	audio8	99	90	82	69	66
9	audio9	90	88	74	60	52
10	audio10	96	91	82	69	61

Fonte: Elaborado pelo autor (2025)

5.3 Resultados SDHASH

O SDHASH demonstrou comportamento notavelmente mais consistente ao longo de toda a faixa de modificação testada, mantendo scores intermediários mesmo em cenários de 40% e 50% de alteração. Em imagens e PDFs, os valores permaneceram frequentemente acima de 50%, evidenciando maior robustez quando comparado ao SSDEEP.

Esse comportamento pode ser atribuído à abordagem fundamentalmente distinta do SDHASH, que utiliza filtros de Bloom para representar características estatísticas do arquivo extraídas por meio de sliding windows de tamanho fixo. Ao contrário do SSDEEP, o SDHASH não depende de pontos de divisão variáveis, mas sim calcula features sobre janelas sobrepostas que percorrem todo o arquivo. Isso significa que alterações localizadas afetam apenas um subconjunto das features, enquanto a maioria permanece inalterada, preservando a medida de similaridade global.

A maior estabilidade observada em PDFs e imagens pode ser explicada pela natureza dos dados binários nesses formatos. Mesmo com alterações de bytes distribuídas ao longo do arquivo, a densidade de features extraídas pelo SDHASH através de suas sliding windows permanece relativamente consistente. Como o algoritmo calcula features sobre janelas sobrepostas de tamanho fixo, a modificação de bytes individuais afeta apenas localmente o conjunto de features, enquanto janelas não sobrepostas às regiões alteradas mantêm suas características originais. Além disso, arquivos binários de imagem e PDF tendem a apresentar padrões de repetição (como pixels adjacentes similares ou estruturas de compressão) que, mesmo parcialmente alterados, ainda geram features com alto grau de sobreposição no filtro de Bloom.

No entanto, a presença de anomalias não lineares, como observado no arquivo audio8 onde o score de 50% de alteração superou o de 40%, revela uma característica importante dos algoritmos baseados em features: a similaridade não é uma função monotônica da quantidade de alteração, mas sim da distribuição espacial dessas modificações. Se as alterações em 50% ocorreram em regiões que já haviam sido parcialmente modificadas em 40%, o conjunto final

de features pode apresentar maior sobreposição. Alternativamente, se as modificações de 40% atingiram features mais representativas (com maior peso na comparação), o score resultante pode ser menor do que em uma alteração de 50% distribuída em features menos críticas.

Este fenômeno é teoricamente previsto na literatura de similarity hashing e está relacionado ao conceito de entropia informacional das regiões modificadas. Arquivos de áudio digital, por exemplo, contêm regiões de alta e baixa entropia (silêncio, ruído, sinal), e alterações em diferentes regiões produzem impactos distintos no conjunto de features extraído.

Desta forma, o SDHASH apresenta-se como uma ferramenta mais robusta para análises que envolvem modificações substanciais ou quando não se conhece a priori a intensidade da manipulação. A não linearidade dos scores, contudo, exige interpretação cuidadosa: um score menor não necessariamente indica mais alteração, mas sim alterações em regiões estruturalmente mais significativas do arquivo

Parâmetros Utilizados no SDhash

A implementação do SDhash analisada utiliza o binário sdhash por meio de um wrapper em Python, que executa comandos específicos para geração, comparação e validação de hashes de similaridade.

Parâmetros de Linha de Comando

Parâmetro -r (Recursive/Read): Utilizado para gerar o hash SDBF de um arquivo. Este parâmetro instrui o sdhash a processar o arquivo de entrada e calcular seu hash de similaridade baseado em Bloom filters. É o parâmetro fundamental para a geração de hashes na função generate().

Parâmetro -o (Output): Define o caminho do arquivo de saída onde o hash SDBF será armazenado. Quando especificado, o sdhash grava o resultado em um arquivo com extensão .sdbf, permitindo posterior comparação e reutilização dos hashes gerados sem necessidade de recalcular.

Parâmetro -c (Compare): Ativa o modo de comparação entre dois arquivos SDBF previamente gerados. Este parâmetro permite calcular a similaridade percentual entre dois conjuntos de dados por meio da comparação de seus Bloom filters. O resultado é retornado no formato arquivo1|arquivo2|percentual.

Parâmetro --validate: Utilizado para validar a integridade e o formato de um arquivo SDBF. Verifica se o arquivo contém um hash válido e se está corretamente formatado conforme as especificações do SDhash.

Formato de Dados

Extensão .sdbf: Todos os hashes gerados pelo SDhash são armazenados em arquivos com extensão .sdbf (Similarity Digest Bloom Filter), que contêm a representação compacta do conteúdo baseada em Bloom filters.

Formato de Saída da Comparação: O resultado da comparação entre dois hashes é retornado no formato arquivo1|arquivo2|similaridade, onde a similaridade é expressa como valor percentual entre 0 e 100.

Parâmetros Implícitos do Algoritmo

Bloom Filter: O SDhash utiliza Bloom filters como estrutura de dados principal para representar o conteúdo dos arquivos. Tipicamente, o algoritmo emprega Bloom filters de 256 bytes para armazenar as características do arquivo.

Tamanho da Janela: O processamento é realizado por meio de janelas deslizantes, geralmente de 64 bytes, que percorrem o arquivo identificando características distintivas baseadas em entropia.

Threshold de Entropia: O SDhash calcula a entropia de cada janela e utiliza um limiar mínimo para determinar quais características são suficientemente distintivas para serem incluídas no hash. Apenas regiões com alta entropia são processadas.

Tamanho Mínimo do Arquivo: O algoritmo requer um tamanho mínimo de arquivo (tipicamente 512 bytes) para gerar um hash válido, garantindo que haja dados suficientes para análise estatística.

Métrica de Similaridade: A similaridade é calculada por meio da interseção dos Bloom filters de dois arquivos, resultando em um valor percentual que indica o grau de sobreposição entre as características identificadas em ambos os arquivos. Valores próximos a 100% indicam alta similaridade, enquanto valores próximos a 0% indicam arquivos distintos.

Tabela 2: Resultados dos testes com SDHASH

Resultados - Testes de similaridade SDHASH

	CSV									
	Nome do arquivo	5%	10%	20%	40%	50%				
1	test1	94.27%	48.06%	46.33%	42.55%	31.72%				
2	test2	97.44%	89.74%	79.49%	64.1%	51.28%				
3	test3	95.34%	90.31%	80.25%	60.14%	50.13%				
4	test4	93.54%	89.48%	76.46%	53.35%	45.57%				
5	test5	91.35%	75.55%	58.93%	54.22%	49.38%				
6	test6	95.03%	90.41%	71.82%	58.84%	48.55%				
7	test7	79.02%	63.54	42.62%	52.96%	35.91%	52.60	59.37		
8	test8	95.35%	90.35%	80.41%	60.4%	50.29%				
9	test9	95.35%	90.32%	80.29%	60.23%	50.19%				

	Imagem - JPG					
	Nome do arquivo	5%	10%	20%	40%	50%
1	img	94.97%	85.74%	77.78%	59.54%	47.38%
2	img2	96.98%	91.88%	78.75%	53.12%	45.62%
3	img3	97.92%	93.62%	79.17%	62.79%	48.94%
4	img4	95.89%	89.04%	79.45%	58.90%	54.79%
5	img5	96.25%	91.25%	79.38%	58.75%	48.12%
6	img6	96.32%	88.42%	77.37%	62.57%	48.11%
7	img7	98.02%	97.03%	85.15%	65.35%	54.46%
8	img8	94.44%	88.89%	79.44%	58.12%	52.78%
9	img9	96.75%	93.50%	83.74%	65.85%	52.85%
10	img10	95.62%	91.19%	81.65%	63.52%	53.12%

	PDF						Testes feitos novamente			
	Nome do arquivo	5%	10%	20%	40%	50%	5%	10%	20%	40%	50%
1	doc1	96.2%	90.11%	79.09%	58.56%	39.92%					
2	doc2	96.08%	90.26%	80.29%	62.98%	51.94%					
3	doc3	91.72%	84.43%	78.44%	58.78%	45.33%					
4	doc4	91.85%	86.39%	76.18%	57.32%	48.77%					
5	doc5	94.54%	88.41%	75.85%	59.04%	46.27%					
6	doc6	52.58%	68.69%	75.21%	43.98%	45.93%	76.26	57.43	65.73	51.38	31.03
7	doc7	95.67%	90.67%	79.70%	58.70%	47.68%					
8	doc8	91.22%	85.33%	74.34%	56.49%	44.46%					
9	doc9	93.42%	89.10%	68.23%	57.89%	37.59%					
10	doc10	94.96%	89.92%	77.13%	38.37%	38.37%				38.37%	38.37%

	Audio						Testes feitos novamente			
	Nome do arquivo	5%	10%	20%	40%	50%	5%	10%	20%	40%	50%
1	audio1	80.17%	59.99%	61.41%	50.46%	31.96%		49.81	61.93		
2	audio2	79.50%	54.67%	55.82%	48.50%	34.94%		57.62	78.67		
3	audio3	61.80%	71.40%	60.14%	47.43%	44.84%					
4	audio4	69.22%	51.51%	65.46%	56.92%	32.99%		59.71	73.71		
5	audio5	84.40%	62.84%	44.04%	31.90%	30.81%					
6	audio6	77.93%	58.08%	51.56%	44.44%	31.21%					
7	audio7	95.38%	83.63%	46.93%	58.72%	44.37%			41.33	52.29	
8	audio8	85.54%	47.26%	73.88%	35.96	47.66%		47.58	79.17	33.67	48.26
9	audio9	65.81%	85.71%	73.49	44.29	31.62%	72.88	69.77			
10	audio10	71.41%	64.21%	76.15%	47.48%	35.57		55.04	55.12		

Fonte: Elaborado pelo autor (2025)

5.4 Resultados TLSH

O TLSH apresentou padrão de degradação variável conforme o tipo de arquivo. Em imagens e PDFs, observou-se queda mais acentuada a partir de 20% de modificação, com transição relativamente abrupta entre alta e média similaridade. Já em arquivos de áudio, os scores demonstraram maior estabilidade, mantendo valores consistentes até 50% de alteração.

A arquitetura do TLSH baseia-se na construção de um hash locality-sensitive que incorpora múltiplas componentes: histogramas de byte values, sliding windows triplets e medidas estatísticas como quartis. A comparação entre hashes não é binária, mas calcula uma distância métrica que pondera as diferenças em cada componente. Essa abordagem multimodal explica por que o TLSH consegue manter certa capacidade de detecção mesmo em altos níveis de modificação.

A diferença de comportamento entre formatos está relacionada à natureza das características capturadas. Arquivos de áudio digital (especialmente em formatos comprimidos) apresentam distribuições de bytes mais uniformes e padrões repetitivos de codificação, resultando em histogramas relativamente estáveis mesmo após modificações no domínio temporal. Já imagens, especialmente em formatos não comprimidos ou com compressão lossless, apresentam maior variabilidade nas distribuições de bytes quando pixels são alterados, impactando mais significativamente os histogramas calculados pelo TLSH.

A anomalia observada na série de testes test5, onde o score de 40% superou o de 20%, pode ser explicada pela natureza probabilística da função de distância do TLSH. As três componentes principais do hash (body digest, Q1/Q2 ratio e checksum) são ponderadas na métrica de distância final. Se as modificações de 20% afetaram desproporcionalmente regiões que influenciam a Q ratio (distribuição de quartis), a distância calculada pode ser maior do que em 40% de alteração distribuída de forma mais homogênea. Isso ocorre porque o TLSH foi projetado para detectar mudanças estruturais significativas, não apenas volume de modificação.

Outro fator relevante é o efeito de quantização nos histogramas. O TLSH utiliza buckets fixos para construir o histograma de features, e dependendo de como os bytes modificados se distribuem entre esses buckets, o impacto na distância final pode ser não linear. Modificações que mantêm os bytes dentro dos mesmos buckets têm impacto menor do que modificações que causam transições entre buckets, mesmo que em menor quantidade.

O TLSH é particularmente eficaz para análise de arquivos de áudio e pode servir como complemento ao SDHASH em investigações que envolvem diversos formatos. A métrica de distância do TLSH fornece informação mais granular do que um score percentual simples, permitindo ao perito forense identificar não apenas se houve modificação, mas também inferir características sobre a natureza dessas alterações (localizadas vs. distribuídas, estruturais vs. superficiais).

Parâmetros Utilizados no TLSH

O TLSH (Trend Micro Locality Sensitive Hash) é uma biblioteca de correspondência fuzzy que gera valores de hash para comparação de similaridade, exigindo um comprimento mínimo de 50 bytes no fluxo de entrada. O fluxo de bytes deve apresentar complexidade suficiente, pois um fluxo de bytes idênticos não gerará um valor de hash.

Parâmetros de Linha de Comando

Parâmetro -r (Recursive): Utilizado para processar arquivos ou diretórios recursivamente, gerando hashes TLSH para cada arquivo encontrado. Essencial para análise em lote de múltiplos arquivos.

Parâmetro -c (Compare): Ativa o modo de comparação entre arquivos, permitindo calcular a distância de similaridade entre dois hashes TLSH. Requer a especificação de um arquivo base seguido dos arquivos ou diretórios a serem comparados.

Parâmetro -T (Threshold): Define o limiar de distância máxima para exibir resultados na comparação. Apenas pares de arquivos com distância menor ou igual ao threshold especificado são exibidos.

Parâmetro -xref: Calcula a distância entre todos os pares de arquivos em um diretório, gerando uma matriz de comparação completa. Pode produzir uma saída extensa, sendo normalmente utilizado com restrição de threshold.

Parâmetro -old: Gera hashes no formato antigo, com 70 caracteres hexadecimais, sem o identificador de versão T1, garantindo compatibilidade com versões anteriores do TLSH.

Estrutura do Hash

O hash gerado contém 35 bytes de dados, exibidos como T1 seguido de 70 caracteres hexadecimais, totalizando 72 caracteres. O identificador T1 foi adicionado como número de versão do hash para permitir a adaptação do algoritmo mantendo a retrocompatibilidade. Os bytes 3, 4 e 5 capturam informações sobre o arquivo como um todo (comprimento, etc.), enquanto os últimos 32 bytes capturam informações sobre partes incrementais do arquivo.

Parâmetros de Configuração de Compilação

O TLSH pode ser compilado para gerar strings de hash de 70 ou 134 caracteres. A versão mais longa foi criada para casos em que o uso de strings de 70 caracteres não é suficiente para a aplicação. O comprimento do hash pode ser aumentado alterando parâmetros de compilação no CMakeLists.txt, aumentando a quantidade de informação armazenada e potencialmente melhorando a precisão na predição de similaridades entre arquivos.

Parâmetros Internos do Algoritmo

Tamanho da Janela: O TLSH utiliza uma janela deslizante de 5 bytes para extrair características do arquivo, baseado no hash Nilsimsa.

Trigramas: São selecionados 6 trigramas dos 10 possíveis de uma janela de 5 bytes (A, B, C, D, E), visando dar representação igual dos bytes na janela

deslizante. Os trigramas 7 a 10 são excluídos porque resultam em contagem duplicada, já que cada trigrama será processado em iteração subsequente.

Função Hash de Pearson: Selecionada como função de mapeamento de buckets devido ao seu histórico e eficácia comprovada. Utilizada para distribuir as contagens de trigramas nos buckets.

Distribuição em Quartis: O TLSH rastreia a distribuição de altura dos buckets de contagem em quartis para melhorar a precisão da comparação. Maior diferença de quartil resulta em score de diferença mais alto.

Score de Similaridade Global: O score de similaridade global distancia objetos com diferença significativa de tamanho. Esta funcionalidade pode ser desabilitada, além de também distanciar objetos com diferentes distribuições de quartis.

Funções de Comparação

Função diff(): Calcula a distância entre dois hashes TLSH, retornando um score onde 0 significa que os objetos são quase idênticos.

Função diffxlen(): Remove o componente de comprimento do arquivo do cabeçalho TLSH da comparação. Se um arquivo repetitivo é comparado a outro com apenas uma instância do padrão, a diferença será maior se o comprimento for considerado. Usando diffxlen, o comprimento é desconsiderado.

Métrica de Distância

A similaridade no TLSH é expressa como um score de distância, onde 0 indica que os objetos são quase idênticos. Scores mais altos indicam maior diferença entre os documentos. Para o hash de 72 caracteres, há uma tabela detalhada de taxas de detecção experimentais e taxas de falsos positivos baseada no threshold.

Requisitos Mínimos

Para um uso conservador, o dado deve conter pelo menos 256 caracteres. Essa restrição garante complexidade suficiente para geração de hash confiável e comparações significativas.

Tabela 3: Resultados dos testes com TLSH

Resultados - Testes de similaridade TLSH

	CSV					
	Nome do arquivo	**5%**	**10%**	**20%**	**40%**	**50%**
1	test1	95,50%	89,00%	65,50%	36,50%	34,50%
2	test2	87,50%	74,50%	59,00%	43,00%	46,00%
3	test3	98,50%	80,50%	49,00%	43,50%	61,00%
4	test4	86,50%	86,50%	38,50%	56,00%	62,50%
5	test5	93,00%	74,50%	38,50%	76,00%	62,50%
7	test6	97,50%	72,50%	53,50%	49,50%	42,50%
8	test7	86,00%	67,50%	41,50%	62,00%	57,50%
9	test8	95,00%	89,00%	41,50%	39,50%	26,50%
10	test9	93,00%	86,50%	71,50%	34,50%	68,5

	Imagem - JPG					
	Nome do arquivo	**5**	**10%**	**20%**	**40%**	**50%**
1	img	81	74	58	41,5	33,5
2	img2	83	70	53,5	25,5	8,5
3	img3	74,5	71,5	55	42	27
4	img4	72,5	75,5	60	47,5	28
5	img5	80,5	77,5	50	49	36
6	img6	77,5	69	65,5	55	41,5
7	img7	78	72,5	57	40,5	32,5
8	img8	74,5	73	51,5	47	28,5
9	img9	75,5	60,5	47	40	17,5
10	img10	78,5	65,5	59,5	30,5	27

	PDF					
	Nome do arquivo	**5%**	**10%**	**20%**	**40%**	**50%**
1	doc1	93,5	77	67,5	58	42
2	doc2	82,5	73,5	75	55,5	48,5
3	doc3	73,5	73	60,5	34,5	35,5
4	doc4	78	77,5	65	49	41,5
5	doc5	80	77,5	69,5	56,5	47
6	doc6	97	97	96	95,5	85
7	doc7	73	74	67	53	39,5
8	doc8	90,5	87	79,5	72,5	73
9	doc9	92,5	89	84,5	71,5	69
10	doc10	90,5	81,5	65,5	48	51,5

	Audio					
	Nome do arquivo	**5%**	**10%**	**20%**	**40%**	**50%**
1	audio1	88	79,5	74	58	59
2	audio2	85,5	82,5	70,5	53	48,5
3	audio3	80	76,5	65	45,5	46,5
4	audio4	84	82,5	72	60	49,5
5	audio5	85,5	77	64,5	55,5	50
6	audio6	84,5	79,5	77,5	63,5	59,5
7	audio7	81,5	75,5	63	57,5	49,5
8	audio8	87	77,5	73	50,5	36
9	audio9	83,5	80,5	69	63	51,5
10	audio10	83,5	79,5	70,5	47,5	36

Fonte: Elaborado pelo autor (2025)

6 CONCLUSÃO

Este trabalho teve como objetivo avaliar o desempenho de três algoritmos de fuzzy hashing — SSDEEP, SDHASH e TLSH — na detecção de alterações em arquivos digitais de diferentes formatos, incluindo CSV, imagens, PDF e áudio. Por meio da implementação de uma prova de conceito em Python, foram realizados testes sistemáticos com alterações graduais de 5% a 50%, permitindo avaliar não apenas a capacidade de detecção de similaridade, mas também compreender os mecanismos teóricos que explicam o comportamento de cada algoritmo.

Os resultados experimentais confirmaram que cada algoritmo apresenta comportamento distinto condicionado por seus fundamentos teóricos. O SSDEEP mostrou-se altamente sensível a pequenas alterações (5-10%) em formatos multimídia, mas sua arquitetura baseada em blocos variáveis limita sua eficácia em modificações acima de 20%, especialmente em arquivos CSV. O SDHASH destacou-se pela robustez em toda a faixa de alteração testada, mantendo scores úteis mesmo com 40-50% de modificação, comportamento consistente com seu mecanismo de extração de features estatísticas sobre janelas sobrepostas. O TLSH apresentou desempenho intermediário, com particular estabilidade em arquivos de áudio devido à uniformidade de suas distribuições de bytes.

Um aspecto fundamental revelado pela pesquisa é que as anomalias não lineares observadas não representam falhas dos algoritmos, mas sim refletem que a similaridade é função da distribuição espacial das modificações, não apenas de seu volume total. Como os algoritmos operam sobre blocos, features ou histogramas estatísticos, a localização e entropia das regiões alteradas têm impacto mais significativo do que o percentual absoluto de bytes modificados.

6.1 Implicações para a Prática Forense Digital

Os resultados obtidos evidenciam que não existe um algoritmo universalmente superior; a escolha adequada depende fundamentalmente do tipo de arquivo sob análise e da natureza das modificações esperadas. Para contextos forenses específicos, as seguintes diretrizes podem ser extraídas:

- Detecção de variantes de malware e análise de código: O SDHASH é recomendado devido à sua robustez em alterações extensas, comum em ofuscação de código malicioso.
- Identificação de adulteração em arquivos multimídia: O SSDEEP é adequado quando as modificações são sutis (até 10-20%), oferecendo alta sensibilidade inicial.
- Análise de arquivos de áudio e agrupamento de arquivos similares: O TLSH apresenta vantagens pela estabilidade de seus histogramas e métrica de distância granular.
- Investigações envolvendo manipulação de dados tabulares (CSV): Todos os algoritmos apresentaram limitações significativas acima de 20% de alteração, sugerindo a necessidade de abordagens complementares ou análise estrutural específica do formato.

Um aspecto fundamental revelado pela pesquisa é que a similaridade não é diretamente proporcional à quantidade de bytes alterados. Como todos os algoritmos operam sobre blocos, features ou distribuições estatísticas — e não por comparação byte a byte —, a localização, distribuição e entropia das regiões modificadas têm impacto mais significativo do que o percentual absoluto de alteração. Essa característica deve ser considerada pelos peritos ao interpretar scores de similaridade em casos reais.

6.2 Contribuições do Trabalho

Este estudo contribui para o campo da forense digital ao:

Fornecer análise comparativa aprofundada dos três principais algoritmos de fuzzy hashing, incluindo explicações teóricas para seus comportamentos observados, não apenas descrições quantitativas dos resultados.

Demonstrar empiricamente as limitações e vantagens de cada algoritmo em diferentes formatos de arquivo, oferecendo subsídios para decisões mais informadas na seleção de ferramentas forenses.

Explicar as anomalias não lineares identificadas nos testes (audio8, test5) com base nos fundamentos teóricos de cada algoritmo, contribuindo para a compreensão mais profunda de seus mecanismos internos.

Evidenciar a importância da distribuição espacial das modificações versus o volume total de alteração, um aspecto frequentemente negligenciado na literatura aplicada.

6.3 Limitações e Trabalhos Futuros

A prova de conceito implementada apresenta limitações que devem ser reconhecidas:

- Escopo limitado de arquivos: Foram testados 10 arquivos de cada tipo (CSV, imagem, PDF, áudio), o que pode não capturar toda a variabilidade de comportamento dos algoritmos em diferentes subtipos e tamanhos de arquivo.
- Padrão de alteração: As modificações foram realizadas de forma contínua em bytes, simulando um tipo específico de manipulação. Outros padrões (alterações pontuais, inserção/deleção de blocos, modificações estruturadas) podem produzir resultados diferentes.
- Ausência de análise temporal: Não foram avaliados os tempos de processamento de cada algoritmo, aspecto relevante para aplicações em larga escala.
- Falta de testes com arquivos comprimidos ou criptografados: Formatos que apresentam alta entropia podem comportar-se de maneira distinta dos testados.

Trabalhos futuros podem expandir esta pesquisa através de:

1. Ampliação do conjunto de testes incluindo maior variedade de formatos (vídeo, arquivos executáveis, documentos Office), tamanhos distintos e subtipos específicos de cada formato.

2. Análise de diferentes padrões de alteração, como inserções localizadas, deleções, transposições e modificações estruturais guiadas por conhecimento do formato.
3. Desenvolvimento de métricas híbridas que combinem os três algoritmos, explorando suas complementaridades para aumentar a precisão e confiabilidade na detecção de similaridade.
4. Investigação de técnicas de machine learning aplicadas aos digests gerados pelos algoritmos, buscando identificar padrões de manipulação específicos.
5. Avaliação de desempenho computacional em cenários de análise forense em larga escala, considerando trade-offs entre precisão e eficiência temporal.
6. Estudo específico sobre detecção de similaridade por contenção, onde um arquivo está parcialmente contido em outro, uma limitação conhecida do TLSH que merece investigação aprofundada.

7 CONSIDERAÇÕES FINAIS

Em síntese, o estudo confirma que a seleção de algoritmos de fuzzy hashing para aplicações forenses deve considerar não apenas a taxa de acerto, mas fundamentalmente os princípios de funcionamento de cada técnica e sua adequação ao tipo de evidência digital sob análise. O SDHASH destacou-se como o mais consistente em alterações intensas, o SSDEEP é recomendado para detecção de pequenas modificações em arquivos multimídia, e o TLSH apresenta vantagens específicas para áudio e análises que requerem métricas de distância granulares.

A integração entre conhecimento teórico dos algoritmos e análise empírica de seus comportamentos mostrou-se essencial para interpretação adequada dos resultados. Peritos forenses devem estar cientes de que scores de similaridade não refletem apenas volume de alteração, mas são mediados pela estrutura dos arquivos, distribuição das modificações e características algorítmicas específicas de cada ferramenta

Esses resultados reforçam a importância de combinar análises quantitativas e qualitativas na avaliação de integridade e similaridade de arquivos digitais, e sugerem que abordagens multi-algoritmo podem oferecer maior robustez e confiabilidade em investigações forenses reais. Este trabalho fornece, assim, subsídios teóricos e práticos para a aplicação consciente e eficaz de técnicas de fuzzy hashing no contexto da perícia digital.

REFERÊNCIAS

ADÃO, John Mário Rinaldini; SILVEIRA, Sirlei Izabel da; SILVEIRA, Daniel Rosa da. **A função do algoritmo hash** MD4. **Revista Científica UNIESP**, Ribeirão Preto, v. 1, n. 1, p. 1-9, 2017. Disponível em: https://uniesp.edu.br/sites/_biblioteca/revistas/20170411124245.pdf. Acesso em: 22 abr. 2025.

AKAD. **A importância da função de hash na segurança da informação**. [S. l.], 14 set. 2023. Disponível em: https://akad.com.br/blog/a-importancia-da-funcao-de-hash-na-seguranca-da-informacao. Acesso em: 21 abr. 2025.

VALLIM, Adriano Penedo de Athayde. **Forense computacional e criptografia**. São Paulo: Editora Senac São Paulo, 2019. (Série Universitária). E-book. ISBN 978-85-396-1254-3. Disponível em: https://www.amazon.com.br/Forense-computacional-criptografia-Universitária-Adriano-ebook/dp/B07X7JFSDM. Acesso em: 22 abr. 2025.

BELAGO. **Análise forense digital:** o que é e como é feita. [S. l.], 14 nov. 2024. Disponível em: https://belago.com/blog/analise-forense-digital-o-que-e/. Acesso em: 23 abr. 2025.

CRIPTOGRAFIA, **saneamento de discos, esteganografia e técnicas antiforenses**: unidade 4 crimes cibernéticos e técnicas forenses. [S. l.: s. n.], [2025?]. Disponível em: https://auxiliar2.telesapiens.com.br/TS/CRIMES_CIBERNETICOS_E_TECNICAS_FORENSES/4/un4/ebook.pdf. Acesso em: 20 mar. 2025.

CRYPTOGRAPHY AND NETWORK SECURITY. **Handbook of applied cryptography**. [S. l.], [2025?]. Disponível em: https://cacr.uwaterloo.ca/hac/. Acesso em: 28 abr. 2025.

DMAPLEARNING. **O que é:** forense digital (digital forensics). [S. l.], 3 ago. 2023. Disponível em: https://dmaplearning.com/glossario/o-que-e-forense-digital-digital-forensics/. Acesso em: 13 abr. 2025.

CÉSAR, Júlio. **Quais são as principais etapas de uma análise forense computacional**. Infosec, [S. l.], 30 mar. 2025. Disponível em: https://www.infosec.com.br/principais-etapas-de-uma-analise-forense-computacional/. Acesso em: 23 abr. 2025.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. FIPS PUB 202:

SHA-3 standard: permutation-based hash and extendable-output functions. [S. l.], 2015. Disponível em:https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. Acesso em: 28 abr. 2025.

PAAR, Christof; PELZL, Jan. **Understanding cryptography:** a textbook for students and practitioners. Berlin: Springer, 2010. Disponível em: https://link.springer.com/book/10.1007/978-3-642-04101-3. Acesso em: 28 abr. 2025.

ALMEIDA, R. N. **Perícia forense computacional:** estudo das técnicas utilizadas para coleta e análise de vestígios digitais. 2011. Trabalho de Conclusão de Curso (Tecnólogo em Processamento de Dados) – Faculdade de Tecnologia de São Paulo, São Paulo, 2011. Disponível em: https://www.fatecsp.br/dti/tcc/tcc0035.pdf. Acesso em: 14 abr. 2025.

PONTES, João Felipe Faria. **Detecção de imagens similares:** aplicabilidade de ferramentas software livre de hash de similaridade de uso geral. [S. l.]: IPOG, 2019. Disponível em: http://assets.ipog.edu.br/wp-content/uploads/2019/12/07015617/joao-felipe-pontes-faria-0198214.pdf. Acesso em: 14 abr. 2025.

PRENEEL, Bart. **Analysis and design of cryptographic hash functions**. 1993. Tese (Doutorado em Ciências Aplicadas) – Katholieke Universiteit Leuven, Leuven, 1993. Disponível em: https://homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf. Acesso em: 03 mai. 2025

ROUSSEV, V. **Data fingerprinting with similarity digests**. In: CHOW, K. P.; SHENOI, S. (org.). Advances in digital forensics VI. IFIP Advances in Information and Communication Technology, v. 337. Berlin: Springer, 2010. p. 207-226. DOI: 10.1007/978-3-642-15506-2_15. Disponível em: https://inria.hal.science/hal-01060620/document. Acesso em: 14 abr. 2025

SANTOS, Hericson dos. **ChaSAM forensics:** uma arquitetura para detecção de imagens na forense computacional utilizando hashes perceptivos. 2024. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024. Disponível em: https://www.teses.usp.br/teses/disponiveis/55/55134/tde-02082024-103051/publico/HericsondosSantos_ME_revisada.pdf. Acesso em: 14 abr. 2025.

SBC SOL. Artigo SBSEG estendido. **Aperfeiçoamento da ferramenta sdhash para identificação de artefatos similares em investigações forenses** [S. l.], 25 out. 2018. Disponível em: https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/4161. Acesso em: 23 abr. 2025.

SBC SOL. **Uso da abordagem hash como ferramenta de análise de similaridade entre imagens**. [S. l.], 9 abr. 2019. Disponível em: https://sol.sbc.org.br/index.php/erbase/article/view/8979/8880. Acesso em: 9 abr. 2025.

SSDEEP PROJECT. **ssdeep:** fuzzy hashing. [S. l.]. Disponível em: https://ssdeep-project.github.io/ssdeep/index.html#docs. Acesso em: 22 abr. 2025.

TORRES, Anderson dos Santos Silva. **Hash perceptivo de imagens e sua aplicação na identificação de cópia de vídeo.** 2019. Dissertação (Mestrado Profissional em Gerência de Redes de Telecomunicações) – Centro de Ciências Exatas, Ambientais e de Tecnologias, Pontifícia Universidade Católica de Campinas, Campinas, 2019. Disponível em: https://repositorio.sis.puc-campinas.edu.br/handle/123456789/15060. Acesso em: 10 abr. 2025.

UNIESP. Revista científica. **A FUNÇÃO DO ALGORITMO HASH MD4** [S. l.], 2017. Disponível em: https://uniesp.edu.br/sites/_biblioteca/revistas/20170411124245.pdf. Acesso em: 22 abr. 2025.

UNIVERSIDADE ESTADUAL DE CAMPINAS. **Aprimoramento de técnicas de hash de similaridade para investigações forenses** – MC2 SBSeg 2016. Campinas, 2016. Disponível em: https://econtents.bc.unicamp.br/eventos/index.php/pibic/article/download/152/105. Acesso em: 8 abr. 2025.

WAGNER, Herbert; MORAIS, V. **Computação forense e suas ferramentas de investigação criminal.** 2020. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Centro Universitário do Rio Grande do Norte, Natal, 2020. Disponível em: http://repositorio.unirn.edu.br/jspui/handle/123456789/731 Acesso em: 24 mar. 2025.