Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

Funções de Hash: Fundamentos e Aplicações em Segurança da Informação

Aline Maria Tenório Gouveia

Trabalho de Graduação

Recife
2024

Universidade Federal de Pernambuco
Centro de Informática

Aline Maria Tenório Gouveia

Funções de Hash: Fundamentos e Aplicações em Segurança da Informação

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Profa. Anjolina Grisi*

Recife
2024

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

Gouveia, Aline Maria Tenório.
Funções de Hash: Fundamentos e Aplicações em Segurança da Informação / Aline Maria Tenório Gouveia. - Recife, 2024.
44 p. : il.

Orientador(a): Anjolina Grisi de Oliveira
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2024.
Inclui referências, apêndices.

1. Funções de Hash. 2. Hash Criptográfico. 3. Integridade de Dados. 4. Autenticidade de Dados. I. Oliveira, Anjolina Grisi de. (Orientação). II. Título.

000 CDD (22.ed.)

Aline Maria Tenório Gouveia

Funções de Hash: Fundamentos e Aplicações em Segurança da Informação

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Recife, Pernambuco, 15 de Outubro de 2024

BANCA EXAMINADORA

_____

Profa. Anjolina Grisi de Oliveira (Orientadora)

Universidade Federal de Pernambuco

_____

Prof. Fernando Maciano de Paula Neto (Examinador Interno)

Universidade Federal de Pernambuco

Ao meu priminho Miguel (Alfinete),
a minha maior motivação para buscar conhecimento.

Agradecimentos

À minha família — meu pai, minha mãe, meu irmão e Leão — agradeço pelo incentivo constante e pela confiança em minhas escolhas.

Agradecimento especial a Bombom e Sofia, pela ajuda na revisão deste texto, e a Beca e Íris, por serem meu ponto de apoio ao longo de tantos anos e ajudarem a encarar as experiências do mundo adulto de forma mais tranquila.

Aos amigos do Centro de Informática, agradeço pelos momentos especiais que compartilhamos nesta jornada acadêmica; sempre torcerei pelo sucesso de cada um de vocês.

Por fim, agradeço à minha orientadora, professora Anjolina, pela paciência e compreensão durante todo o processo de elaboração deste trabalho, e ao professor Ruy, por dedicar seu tempo à revisão final.

Life isn't about waiting for the storm to pass...
It's about learning to dance in the rain.

—VIVIEN GREENE

Resumo

As funções de hash, também chamadas de funções de resumo ou dispersão, são uma categoria de funções que mapeiam uma entrada de tamanho arbitrário para uma cadeia de tamanho fixo previamente definido. Em contextos práticos, o valor gerado, o hash, é uma cadeia de bits que representa diferentes tipos de informações. Este trabalho investiga essas funções, explorando seus fundamentos para oferecer uma compreensão abrangente do tema e apresentando conceitos básicos. São definidas tanto as funções de hash não baseadas em chave quanto as famílias de funções de hash que dão origem às funções baseadas em chave. Também são abordados os requisitos de segurança, que incluem os principais problemas utilizados para analisar e medir a robustez de uma função: pré-imagem, segunda pré-imagem e colisão. A pesquisa ainda analisa as funções de hash criptográficas, primeiramente apresentando uma taxonomia que as divide em Funções de Códigos de Autenticação de Mensagem (*Message Authentication Codes* - MACs) e Códigos de Detecção de Modificação (*Modification Detection Codes* - MDCs), e, em seguida, explorando suas principais famílias com base no padrão SHA (*Secure Hash Algorithm*). Além disso, o estudo destaca várias aplicações dessas funções, com ênfase no seu papel crucial em garantir a integridade e autenticidade em ambientes digitais. A principal contribuição deste trabalho reside na introdução de conhecimentos fundamentais que sustentam a utilização de funções de hash.

Palavras-chave: Hash, SHA, Segurança da Informação, Integridade, Autenticidade

Abstract

Hash functions are a category of functions that map an input of arbitrary size to a fixed-length output. In practical contexts, the generated value, the hash, is a string of bits that represents different types of information. This paper investigates these functions, exploring their fundamentals to provide a comprehensive understanding of the subject while introducing basic concepts. Both unkeyed hash functions and hash function families, which give rise to key-based hash functions, are defined. Security requirements are also addressed, including key challenges used to assess the robustness of a function: preimage, second preimage, and collision. The research further describes cryptographic hash functions, first presenting a taxonomy that divides them into Message Authentication Codes (MACs) and Modification Detection Codes (MDCs), and then exploring their main families based on the Secure Hash Algorithm (SHA) standard. Additionally, the study highlights various applications of these functions, emphasizing their crucial role in ensuring integrity and authenticity in digital environments. The main contribution of this work lies in introducing fundamental knowledge that supports the use of hash functions.

Keywords: Hash, SHA, Information Security, Integrity, Authenticity

Sumário

Lista de Figuras

Lista de Acrônimos

CRHF Collision-Resistant Hash Function

DES Data Encryption Standard

FIPS Federal Information Processing Standards

HMAC Hash-based Message Authentication Code

HTTPS Hypertext Transfer Protocol Secure

IV Initialization Vector

IPsec Internet Protocol Security

KWIC Key Word in Context

MAC Message Authentication Code

MDC Modification Detection Code

MD Message Digest

MIC Message Integrity Code

NIST National Institute of Standards and Technology

NSA National Security Agency

OWHF One-Way Hash Function

OTP One-Time Password

PRF Pseudo-Random Function

RSA Rivest-Shamir-Adleman

SHS Secure Hash Standard

SHA Secure Hash Algorithm

SSL Secure Sockets Layer

TLS Transport Layer Security

WPA Wi-Fi Protected Access

CAPÍTULO 1

Introdução

1.1 Contextualização e Motivação

O conceito de *hashing* refere-se à aplicação de uma função de hash a uma entrada de tamanho arbitrário, gerando uma saída de tamanho fixo e pré-definido pela função. Embora as chamadas funções de hash criptográficas sejam um caso particular das funções de hash, o segundo termo é frequentemente utilizado como sinônimo do primeiro — essa utilização ocorre especialmente em contextos onde se assume o uso exclusivo dessa categoria.

De forma análoga, o valor de hash é identificado, especialmente em textos relacionados à criptografia, por uma variedade de sinônimos, conforme destacado por Preneel [19]:

> "[...] o resultado da função de hash recebeu uma ampla variedade de nomes na literatura criptográfica: hashcode, total de hash, resultado de hash, impressão, checksum (criptográfico), compressão, codificação comprimida, selo, autenticador, tag de autenticação, impressão digital, chave de teste, condensação, Código de Integridade de Mensagem (MIC), resumo de mensagem, etc."
>
> *"[...] the result of the hash function has been given a wide variety of names in the cryptographic literature: hashcode, hash total, hash result, imprint, (cryptographic) checksum, compression, compressed encoding, seal, authenticator, authentication tag, fingerprint, test key, condensation, Message Integrity Code (MIC), message digest, etc."*

Essa diversidade de termos surge da vasta aplicação dessas funções, de modo que diferentes contextos nomeiam o valor de hash conforme suas necessidades específicas. Além da utilização em estruturas de dados, onde servem como base para a construção de tabelas de hash e otimização de consultas em bancos de dados, essas funções são fundamentais em contextos em que a garantia de unicidade e integridade das informações é crítica.

1.1.1 Histórico

O termo "hash" tem origem no verbo francês *hacher*, que significa "cortar". Assim como palavras podem ser desmembradas para serem ressignificadas, as funções de hash transformam informações.

No contexto da computação, o termo foi popularizado em 1968 por Robert Morris, em seu artigo intitulado "Técnicas de Armazenamento Distribuído" (*Scatter Storage Techniques*) [16].

Neste trabalho, Morris apresentou discussões sobre tabelas de hash e técnicas de armazenamento distribuído, utilizando funções de hash para otimizar a eficiência de busca e armazenamento de dados.

Entretanto, ideias subjacentes ao *hashing* já existiam anteriormente. Hans Peter Luhn, um dos pioneiros no campo da ciência da informação, fez contribuições significativas que podem ser consideradas precursoras das aplicações atuais das funções de hash [23].

Em 1954, Luhn registrou uma patente nos Estados Unidos para um "Computador para Verificação de Números", que influenciou diretamente o desenvolvimento de seu conhecido "Algoritmo de Luhn", patenteado em 1960. Este algoritmo, também conhecido como "Mod 10", é muito utilizado atualmente como verificador da validade de cartões de crédito. Ele consiste em uma sequência de operações matemáticas aplicadas aos números do cartão, gerando um código final que serve como verificador da sua validade.

Ainda em 1958, durante a Conferência Internacional de Informação Científica, Luhn também apresentou um sistema automatizado da IBM capaz de gerar índices de documentos, baseado em seu algoritmo conhecido como KWIC (*Key Word in Context*) [14]. O sistema organizava entradas com base em palavras-chave extraídas de seus textos identificadores, como títulos de artigos técnicos, excluindo termos irrelevantes, como conectivos.

O KWIC permitia localizar informações rapidamente ao agrupar as entradas alfabeticamente por suas palavras-chave, facilitando a inclusão de novas entradas no índice e reduzindo a subjetividade das classificações manuais, que costumavam depender da categorização por temas. Na abordagem de Luhn, as entradas tinham um lugar definido, determinado unicamente pelo texto que as identificava.

O sistema já introduzia uma ideia de limitação na cadeia identificadora, ao exibir até 60 de seus caracteres, o que poderia resultar na truncação de algumas palavras, dependendo do tamanho original.

A Figura 1.1 é um exemplo de índice disponibilizado por Luhn, ilustrando seu sistema. As palavras-chave dos títulos são organizadas em ordem alfabética, formando uma coluna central; as palavras anteriores a ela aparecem à esquerda e as posteriores à direita, em um formato "rotacionado", ou seja, as palavra-chave dos títulos ocupam uma posição de destaque, com as demais palavras deslocadas para os lados. Como um título pode ter várias palavras-chave, também pode ser repetido no índice — por exemplo, o título "Gamma Rays in Ge 72" aparece tanto sob "Gamma"quanto sob "Ge". Por fim, cada linha é associada a um número (ou referência) que identifica o documento correspondente.

Essas contribuições estabeleceram as bases para o entendimento de que o pré-processamento de informações pode otimizar a consulta e validação de dados.

A evolução das funções de hash modernas, especialmente aquelas utilizadas em segurança digital, teve início em 1976, marcada pela publicação do artigo "Novas Direções em Criptografia" (*New Directions in Cryptography*), de Diffie e Hellman [12]. Este trabalho introduziu a criptografia de chave pública e destacou a importância das funções unidirecionais (*one-way functions*), projetadas para serem fáceis de computar em uma direção, mas difíceis de reverter. Essa característica é desejável também para as funções de hash, e nomeia uma das suas classificações (ver definição 3.1.2).

No final da década de 1970, surgiram as primeiras definições e propostas que levaram ao

Keyword-in-Context Bibliographical Index

	EXCITATION OF PROTONS IN HELIUM II B	0011
OF ATOMIC AND MOLECULAR	EXCITATION BY A TRAPPED-ELECTRON ME	0150
THERMAL	EXCITATIONS IN LIQUID HE3.	1465
ENERGIES OF GROUND AND	EXCITED NUCLEAR CONFIGURATIONS IN TH	0452
	EXCITED STATES OF V51 AND CR53.	1691
4-PLUS	EXCITED STATE IN OSMIUM-188.	1717
NTERNAL PHOTOEFFECT AND	EXCITON DIFFUSION IN CADMIUM AND ZIN	0123
OF THE CONTRIBUTION OF	EXCITONS TO THE COMPLEX DIELECTRIC	1555
THERMAL	EXPANSION OF SOME CRYSTALS WITH THE	0136
ENERGY LEVELS IN	F18 FROM THE N14/ALPHA,ALPHA/N14 AND	0547
ON FROM AL27-PLUS-P AND	F19-PLUS-P.	0239
TIC MEASUREMENTS OF THE	FE-CR SPINELS.	1603
BARIUM	FERRATE III.	0326
MAGNETOSTATIC MODES IN	FERRIMAGNETIC SPHERES.	0059
NICKEL-IRON	FERRITE.	0397
TRANSITION TO THE	FERROELECTRIC STATE IN BARIUM TITANA	0413
SUPERCONDUCTIVITY AND	FERROMAGNETISM IN ISOMORPHOUS COMPOU	0089
INTERPLANETARY MAGNETIC	FIELD AND ITS CONTROL OF COSMIC-RAY	0589
MAGNETIC	FIELD DEPENDENCE OF ULTRASONIC ATTEN	0080
RELATIVISTIC	FIELD THEORY OF UNSTABLE PARTICLES.	0283
QUANTUM	FIELD THEORIES WITH COMPOSITE PARTIC	0669
A GENERALLY CONVARIANT	FIELD THEORY.	1826
AND SURFACE STATES FROM	FIELD-INDUCED CHANGES IN SURFACE REC	0369
NGULAR DISTRIBUTIONS IN	FISSION INDUCED BY ALPHA PARTICLES.	0536
UTRON CROSS SECTIONS OF	FISSIONABLE NUCLEI.	0203
AL COSMIC-RAY INTENSITY	FLUCTUATIONS OBSERVED AT SOUTHERN ST	1798
	FLUX OF COSMIC-RAY PARTICLES WITH Z-	0597
NEUTRINO CORRELATION IN	FORBIDDEN BETA DECAY.	0244
	FOURIER COEFFICIENTS OF CRYSTAL POTE	0073
RVATION IN THE DECAY OF	FREE AND BOUND LAMBDA PARTICLES.	0605
STEADY-STATE	FREE PRECESSION IN NUCLEAR MAGNETIC	1693
	FREQUENCY SHIFT OF THE ZERO-FIELD HY	0449
DECAY OF	GADOLINIUM-159.	0262
	GAMMA RADIATION FROM AL27-PLUS-P AND	0239
ECTIONAL CORRELATION OF	GAMMA RAYS IN GE72.	0229
CISION DETERMINATION OF	GAMMA RAYS FOLLOWING P,P-PRIME-GAMMA	0532
	GAMMA-RAY THRESHOLD METHOD AND THE O	0461
P/S32 AND S32/P,P-PRIME	GAMMA/S32.	1702
ONSTANT OF YTTRIUM IRON	GARNET AT 0 DEG K.	0395
LORENTZIAN	GAS AND HOT ELECTRONS.	1567
TIBILITY OF AN ELECTRON	GAS AT HIGH DENSITY.	0328
UCTIVITY OF AN ELECTRON	GAS IN A GASEOUS PLASMA.	0001
OF AN ELECTRON GAS IN A	GASEOUS PLASMA.	0001
DUCED BY VARIOUS BUFFER	GASES.	0449
BUFFER	GASES.	0450
IONIZED	GAS.	1441
EZORESISTANCE IN N-TYPE	GA,AS.	1533
IN ELECTRON-IRRADIATED	GE AT 80 DEG K.	0362
LATION OF GAMMA RAYS IN	GE72.	0229
NERAL RELATIVITY AS THE	GENERATORS OF COORDINATE TRANSFORMAT	0287
ETORESISTANCE IN N-TYPE	GERMANIUM AT LOW TEMPERATURES.	0317
CONDUCTION ELECTRONS IN	GERMANIUM.	0298
IATIVE RECOMBINATION IN	GERMANIUM.	0330
PARTICLES IN LINEARIZED	GRAVITATIONAL THEORY.	0674

Figura 1.1 Exemplo de índice elaborado com o sistema KWIC [14].

que hoje conhecemos como funções de hash criptográficas, um subconjunto das funções de hash com propriedades adicionais que garantem segurança em contextos criptográficos. Dentre essas contribuições, destacam-se Rabin, que projetou uma função de hash de 64 bits baseada no DES (*Data Encryption Standard*); Yuval, que demonstrou como encontrar colisões (ver Seção 2.2) em tempo proporcional a $2^{n/2}$ utilizando o paradoxo do aniversário[1]; e Merkle, que introduziu os conceitos de resistência à colisão e à pré-imagem. O conceito de colisão, formalizado por Damgård [9] em 1987, será apresentado na Seção 2.3.

Mais recentemente, em 2004, Rogaway e Shrimpton estudaram as relações entre resistência à colisão e resistência à pré-imagem, destacando a importância de funções de hash que garantem a segurança contra ataques.

[1] O paradoxo do aniversário ilustra como a probabilidade de colisões aumenta rapidamente em conjuntos pequenos, fazendo uma analogia com os dias de aniversário de um grupo. Katz e Lindell [13] exploram esse conceito e ataques relacionados.

Esse panorama histórico, abordado por Preneel em seu trabalho sobre os primeiros 30 anos das funções hash criptográficas [20], demonstra que as crescentes demandas de segurança, provenientes de suas diversas aplicações, impulsionaram avanços significativos que resultaram na variedade de projetos de funções hash que conhecemos hoje.

1.2 Objetivos

O objetivo geral deste trabalho é oferecer uma visão abrangente sobre as funções de hash e seus conceitos fundamentais, com ênfase em sua segurança e aplicações na área de segurança da informação.

Os objetivos específicos são:

I. Compreender as definições básicas das funções de hash e suas principais classificações.

II. Compreender requisitos considerados em análises de segurança das funções de hash.

III. Explorar o uso de funções de hash criptográficas, destacando sua importância e os critérios para a definição dessas funções.

IV. Estudar exemplos de aplicações práticas das funções de hash.

Ao final, espera-se que o estudo contribua para o entendimento das funções de hash como pilares essenciais na proteção da integridade e autenticidade dos dados no ambiente digital.

1.3 Estrutura do Trabalho

O presente texto está organizado nos seguintes capítulos:

- **Capítulo 1**, que expõe a motivação do trabalho, diretamente ligada à importância histórica do tema.
- **Capítulo 2**, em que são introduzidos os conceitos fundamentais para a compreensão de textos relacionados às funções de hash, incluindo a definição geral e as famílias de funções baseadas em chave. Também são abordados os problemas fundamentais para as provas de segurança dessas funções — o problema da pré-imagem, o problema da segunda pré-imagem e o problema da colisão, sendo este último detalhado em uma seção própria. Por fim, apresentamos uma descrição da noção de ataques, um conceito essencial para entender a evolução das funções de hash.
- **Capítulo 3**, em que definimos as funções de hash que atendem a critérios de segurança mais rigorosos, conhecidas como funções de hash criptográficas. Também descrevemos suas principais famílias.

- **Capítulo 4**, em que discutimos as diversas aplicações das funções de hash, evidenciando sua presença e relevância. Esta seção visa conectar os conceitos apresentados ao longo do texto, enfatizando a importância das funções de hash na segurança da informação.
- **Capítulo 5**, que conclui este trabalho de graduação, apresentando notas finais após a realização do estudo.

CAPÍTULO 2

Fundamentos

2.1 Definições Básicas

Uma função de hash é um tipo de função projetada para mapear entradas de tamanho variável para saídas de tamanho fixo. Além disso, é esperado que essas saídas sejam uniformemente distribuídas.

Nas definições a seguir, assim como em todo o restante deste trabalho, utilizaremos com frequência o termo "mensagens", que refere-se às entradas esperadas para os algoritmos. Em essência, uma mensagem é uma sequência de bits que representa alguma informação, abrangendo diferentes tipos de dados, como arquivos ou textos. De forma similar, o termo "cadeias" refere-se a cadeias de bits.

As funções de hash podem ser classificadas em duas categorias principais: não baseadas em chave e baseadas em chave. As não baseadas em chave, frequentemente referidas simplesmente como "funções de hash", podem ser definidas da seguinte forma:

Definição 2.1.1. *Seja S o conjunto de todas as possíveis mensagens e T um conjunto finito de cadeias de tamanho fixo, tal que $|S| > |T|$. Uma função $H : S \to T$ é considerada uma função de hash se, para cada elemento $s \in S$, existe um elemento $t \in T$ tal que $H(s) = t$. Assim, H mapeia cada mensagem de entrada em uma cadeia de tamanho fixo.*

Por outro lado, as funções de hash baseadas em chaves podem ser entendidas como parte de uma família de funções de hash. Cada chave — uma cadeia específica de caracteres — gera uma função de hash única. A definição de família de funções de hash é a seguinte:

Definição 2.1.2. *Sejam S e T os conjuntos definidos anteriormente, e K um conjunto finito de chaves geradas por um algoritmo intrínseco à família. Uma família de funções de hash H é uma função que associa cada par $(k,s) \in K \times S$ a um valor de hash em T, expressa como $H_k : S \to T$. Isso significa que, para cada chave k gerada, existe uma função de hash correspondente H_k mapeia cada mensagem de entrada em uma cadeia de tamanho fixo.*

Adicionalmente, uma função de hash não baseada em chave pode ser interpretada como uma família de funções de hash em que existe apenas uma única chave possível, ou seja, quando $|K| = 1$. Algumas provas consideram exclusivamente o conceito de família e, consequentemente, funções baseadas em chave, para alcançar uma definição matematicamente precisa que engloba ambas as categorias [11].

Esta generalização é importante porque funções de hash não baseadas em chave são também muito comuns em aplicações práticas, especialmente nas que fazem uso das funções de hash criptográficas, que são abordadas no Capítulo 3.

2.2 O Problema da Colisão

Analisando as definições 2.1.1 e 2.1.2, percebemos que os conjuntos S e T têm uma relação interessante: $|S| > |T|$. Isso significa que a função de hash $H : S \to T$ não pode ser injetora, pois múltiplas entradas podem resultar na mesma saída. No contexto das funções de hash, a ocorrência desse evento é chamada de colisão.

Fazendo uma analogia com o Princípio da Casa dos Pombos, concluímos que as colisões são inevitáveis: o domínio das entradas (pombos) é potencialmente ilimitado em comparação com o conjunto de saídas finito (casas), já que existem menos casas do que pombos para serem alocados.

Nesse sentido, historicamente, definiu-se que, para que uma função H seja considerada resistente a colisões, é suficiente que, em termos de complexidade de tempo e espaço, seja computacionalmente inviável encontrar uma colisão. O termo "computacionalmente inviável" indica que o custo, medido tanto pela quantidade de memória utilizada quanto pelo tempo de execução, é finito, mas extremamente alto [12].

Por razões de segurança, se uma colisão for encontrada em uma função de hash, ela é considerada "quebrada", já que esse evento deveria ser extremamente difícil de ocorrer [24]. Isso destaca um dos principais desafios na criação de funções de hash: como projetá-las de modo a reduzir significativamente a probabilidade de colisões.

2.2.1 Funções de Hash Universais

As funções de hash universais foram uma das primeiras tentativas de minimizar colisões. O conceito de universalidade é uma maneira formal de definir limites probabilísticos de modo a garantir que as funções de hash de uma família não favoreçam nenhuma entrada ao gerar um hash, ajudando a controlar e minimizar colisões.

Esse princípio é amplamente aplicado em funções de hash utilizadas para construir tabelas de hash, em que geralmente precisa-se garantir que as colisões ocorram com uma probabilidade suficientemente baixa. Contudo, com o aumento das exigências de segurança e o avanço do poder computacional, ficou evidente que as funções de hash universais não eram suficientes para proteger dados críticos. Isso levou ao desenvolvimento de funções de hash com características de segurança adicionais, conhecidas como funções de hash criptográficas (ver Capítulo 3).

Partindo da definição 2.1.2 e da proposta de Stinson e Paterson [24] sobre o conceito de uma família de funções de hash "fortemente universal", podemos definir esse limite da seguinte maneira.

Definição 2.2.1. *Uma família de funções de hash é denominada "fortemente universal" se, para quaisquer duas entradas distintas s e s' em S, e para quaisquer dois valores de hash t e t' em T, o número de chaves k em K que satisfazem $H_k(s) = t$ e $H_k(s') = t'$ deve ser igual ao número total de chaves em K. A seguinte expressão formaliza essa condição:*

$$|\{k \in K : H_k(s) = t, H_k(s') = t'\}| = |K|.$$

2.3 Noções de Segurança

A avaliação da segurança das funções de hash geralmente envolve o uso de métricas que quantificam, em termos de complexidade computacional, a eficácia da função em atender problemas de segurança específicos. Isso se deve ao fato de que, em implementações reais, é improvável que todos os requisitos esperados sejam atendidos por completo.

Três problemas principais são frequentemente abordados nas provas e análises de segurança dessas funções. Esse problemas serão descritos a seguir.

- **O problema da pré-imagem**: Refere-se à dificuldade de descobrir a entrada original s a partir de um hash já calculado $H(s)$. Isso significa que, dado um hash h, deve ser difícil encontrar um valor s tal que $H(s) = h$.

  O problema da pré-imagem é equivalente a atender à propriedade de ser unidirecional (*one-way*).

- **O problema da segunda pré-imagem**: Refere-se à dificuldade em descobrir uma segunda entrada s' diferente de s que resulte no mesmo hash. Portanto, dado um par $(s, H(s))$, deve ser difícil encontrar um s' tal que $H(s') = H(s)$.

  O problema da segunda pré-imagem é equivalente a atender à propriedade de resistência fraca à colisões (*weak collision resistance*).

- **O problema da colisão**: Refere-se à dificuldade em encontrar entradas s e w (com $s \neq w$) que satisfaçam $H(s) = H(w)$.

  O problema de colisão é equivalente a atender à propriedade de resistência forte à colisões (*strong collision resistance*).

Se uma função de hash é resistente ao problema da colisão, ela também deve ser resistente ao da segunda pré-imagem, uma vez que este é um caso específico do primeiro. No entanto, essa inferência pressupõe funções de hash bem projetadas — as que realizam alguma compressão não trivial de seus dados [11]. Por exemplo, considere a função identidade, $I(s) = s$. Embora essa função não apresente colisões — pois cada entrada se mapeia a ela mesma — ela não oferece resistência à pré-imagem ou à segunda pré-imagem, uma vez que qualquer valor de saída pode ser facilmente revertido para a entrada correspondente.

Além dos problemas fundamentais, existem outros aspectos relevantes, que Menezes et al. [15] se referem como "propriedades de validação de funções de hash" (*certificational hash function properties*)[1]. Um exemplo importante é o efeito avalanche, que se relaciona à distribuição uniforme das entradas no espaço de saída e ao conceito de *completude* de uma operação que lida com bits. A ideia é que uma pequena alteração na entrada cause uma grande mudança na saída, tornando esta realmente dependente de todos os bits originais.

Os autores mencionam um critério rigoroso proposto por Webster e Tavares [27]: "Para que uma determinada transformação apresente o efeito avalanche, em média, metade dos bits de

[1] A tradução utiliza o termo "validação" por refletir melhor o sentido de *certificational* no contexto da expressão original.

saída deve mudar sempre que um único bit de entrada for invertido.". Por exemplo, considere uma mensagem m e sua variante m', com apenas um caractere de diferença. O objetivo é que essa mínima alteração entre m e m' cause uma mudança significativa e imprevisível nos respectivos hashes, tornando impossível prever o hash de uma mensagem mesmo conhecendo o da outra.

Outro aspecto a ser considerado é a necessidade de que o algoritmo não seja excessivamente rápido em alguns contextos. Em sistemas de autenticação por senha, por exemplo, essa consideração ajuda a prevenir ataques de força bruta — o algoritmo deve ser ágil o suficiente para garantir eficiência, mas não tão rápido a ponto de facilitar a execução esses ataques.

Um conceito relevante nesse contexto é o fator de trabalho (*work factor*), que adiciona intencionalmente complexidade ao algoritmo ao aumentar a quantidade de iterações necessárias em sua computação [10].

2.3.1 Modelo do Oráculo Aleatório

Para definir uma função de hash "ideal", utiliza-se o conceito do Modelo do Oráculo Aleatório. Essa abordagem teórica, embora sem reivindicações práticas, fornece uma metodologia robusta para o projeto e validação de esquemas criptográficos, por exemplo, sendo amplamente reconhecida na literatura como uma base para a construção de provas. Katz e Lindell [13] ressaltam que a segurança de um esquema criptográfico está intimamente ligada à dificuldade de encontrar colisões; ou seja, se a função de hash for considerada segura, então o esquema como um todo também o será.

Introduzido por Bellare e Rogaway [8] em 1993, o modelo representa uma função de hash ideal como uma "caixa preta". Nesse contexto, não são disponibilizadas fórmulas ou algoritmos para calcular os valores da função H; a única maneira de determinar $H(s)$ é por meio de consultas ao oráculo. Essa abordagem garante que, mesmo com acesso a hashes de outras instâncias, o hash de uma entrada específica só pode ser obtido invocando o oráculo, preservando assim a segurança e a imprevisibilidade da função de hash.

Stinson e Paterson [24] analisam os três problemas principais de análise de segurança mencionados anteriormente, relacionando-os às funções de hash na concepção do oráculo. Com base em suas provas (que sugerimos como leitura adicional), a conclusão é que o problema da colisão é o mais simples de ser quebrado. Na apresentação de medidas específicas, como será demonstrado na Seção 3.2, essa noção se tornará mais clara, pois geralmente a quantidade de tentativas necessárias para quebrar o problema da colisão é menor.

2.4 Noções sobre Ataques

Na literatura, ataques são frequentemente descritos como experimentos exaustivos realizados para alcançar objetivos específicos, geralmente de natureza maliciosa. No contexto deste trabalho, as propriedades de segurança estão intrinsecamente relacionadas à probabilidade de sucesso desses experimentos. Por exemplo, a resistência a colisões pode ser definida como a capacidade, ou facilidade, de um adversário encontrar uma colisão em uma instância de uma família de funções de hash H.

Dessa forma, a evolução dessas funções está diretamente relacionada ao conceito de *adversários*. No contexto criptográfico, além da interação entre um emissor e um receptor, considera-se a presença de um adversário que tenta manipular os dados intencionalmente. Para as funções de hash, essa ameaça se manifesta na tentativa de encontrar colisões, evidenciando a necessidade de segurança em diversas aplicações.

Nesse sentido, funções de hash não baseadas em chave são conceitualmente mais vulneráveis a ataques [13]. Para ilustrar, considere um ataque teórico que busca desenvolver um algoritmo capaz de retornar um par de entradas que produzem o mesmo valor de hash. Se esse ataque conseguir encontrar uma colisão em uma função U (não baseada em chave), o algoritmo pode ser ajustado para retornar, em tempo constante, esse par já conhecido ao receber a função U.

Em contrapartida, em uma função K (baseada em chave), a presença de uma chave secreta altera os valores de saída de K. Assim, mesmo que os valores de entrada sejam idênticos, não seria possível ajustar o algoritmo para gerar uma colisão em K em tempo constante.

A forma mais elementar de ataque é a força bruta, que consiste em avaliar mensagens e gerar seus hashes correspondentes. Se considerarmos que os valores de hash têm 128 bits de tamanho, o número total de cadeias possíveis é 2^{128}. Portanto, a quantidade de tentativas (ou passos) necessárias para encontrar uma colisão seria 2^{128}.

Por outro lado, ao usar um ataque de aniversário, que se baseia no paradoxo do aniversário, espera-se encontrar uma colisão após cerca de 2^{64} tentativas, a raiz quadrada de 2^{128} [24][10]. Essa diferença significativa ilustra a eficiência de métodos de ataque que exploram propriedades matemáticas.

Adversários estão constantemente em busca de maneiras de explorar vulnerabilidades. O criptógrafo Bruce Schneier publicou em 2004 que: "Mas há um velho ditado dentro da NSA (Agência de Segurança Nacional dos Estados Unidos): 'Os ataques sempre melhoram; nunca pioram.'"[22].

CAPÍTULO 3

Funções de Hash Criptográficas

Uma função de hash H é considerada criptográfica quando atende com relativa dificuldade um subconjunto dos três principais problemas descritos na Seção 2.3: pré-imagem, segunda pré-imagem e colisão. Embora seu uso seja predominante na área de criptografia, o termo "funções de hash criptográficas" está mais relacionado ao sentido de serem consideradas seguras o suficiente para serem aplicadas em contextos criptográficos, como também a qualquer contexto que envolva dados sensíveis.

A construção de algoritmos seguros criptograficamente geralmente se baseia em problemas matemáticos com dificuldade computacional comprovada, como a fatoração de números primos; Menezes et al. [15] detalham a geração de números primos válidos para esse propósito. Katz e Lindell [13] complementam essa perspectiva ao afirmar que uma função de hash pode ser considerada resistente a colisões se o número de tentativas necessário para encontrá-las for inviável para qualquer algoritmo probabilístico em tempo polinomial. Assim, a dificuldade em encontrar colisões reflete a ausência de algoritmos eficientes capazes de descobri-las em um tempo razoável, assim como acontece com os problemas matemáticos adotados.

Neste capítulo, a Seção 3.1 apresentará uma proposta de classificação dessas funções de hash. Por fim, a Seção 3.2 descreverá as funções consideradas aprovadas conforme as diretrizes do NIST, que desempenha um papel crucial na regulamentação e padronização de funções de hash criptográficas.

3.1 Classificação de Funções de Hash Criptográficas

Para facilitar a compreensão das propriedades das funções de hash criptográficas, este trabalho adotará a taxonomia proposta por Preneel [19] (também de acordo com Menezes et al. [15]). A divisão dessa taxonomia está ilustrada na Figura 3.1, e as definições das classes propostas foram adaptadas do mesmo autor.

Para padronizar a representação nas definições, a função de hash será denotada por H, enquanto a entrada, ou seja, a mensagem a ser processada, será representada por s. A saída correspondente a s sob a função de hash H será indicada por $H(s)$, e a chave secreta será representada por k.

A primeira condição em cada definição é uma extensão do princípio de Kerckhoffs, que afirma que a segurança de um sistema criptográfico deve depender apenas da escolha das chaves, presumindo acesso público a todos os outros componentes.

Os tamanhos estabelecidos para o valor de hash, conforme indicado pelo autor de referência, são fundamentados em análises de segurança que consideram a resistência a ataques

específicos.

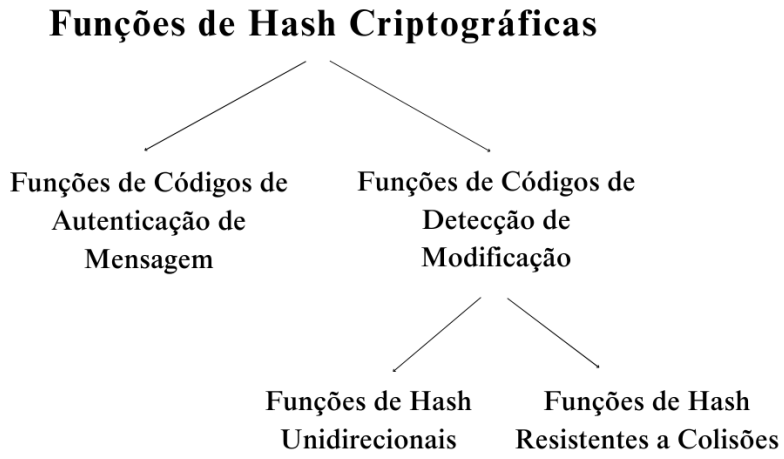

Figura 3.1 Taxonomia das funções de hash criptográficas, adaptada de Preneel [19].

3.1.1 Funções de Códigos de Autenticação de Mensagem (*Message Authentication Codes* - MACs)

Frequentemente chamadas de "funções de hash com chave", as funções de MAC dependem de uma chave secreta compartilhada entre o remetente e o destinatário. Uma aplicação direta desta classe será abordada na Seção 4.1. Segue a definição:

Definição 3.1.1. *Uma função de MAC satisfaz as seguintes condições:*

- *A descrição de H deve ser publicamente conhecida, e a única informação secreta reside na chave k.*
- *O argumento s pode ter comprimento arbitrário, e o resultado $H(k,s)$ tem um comprimento fixo de n bits (com $n \geq 32 \ldots 64$).*
- *Dado H, s e k, o cálculo de $H(k,s)$ deve ser "fácil".*
- *Dado H e s, é "difícil" determinar $H(k,s)$ com uma probabilidade de sucesso "significativamente maior" do que $1/2^n$. Mesmo quando um grande conjunto de pares $\{s_i, H(k,s_i)\}$ é conhecido, em que os s_i foram selecionados pelo adversário, é "difícil" determinar a chave k ou calcular $H(k,s')$ para qualquer $s' \neq s_i$.*

3.1.2 Funções de Códigos de Detecção de Modificação (*Modification Detection Codes* - MDCs)

As funções de MDC têm o objetivo primário de gerar o hash de uma mensagem, utilizado para verificar a integridade dos dados. Elas desempenham o papel de auxiliar na detecção de alterações, garantindo que os dados transmitidos ou armazenados não foram modificados indevidamente.

As MDCs são funções de hash sem chave e podem ser divididas em dois tipos principais: as OWHFs e as CRHFs.

3.1.2.1 Funções de Hash Unidirecionais (*One-Way Hash Functions* - OWHFs)

Definição 3.1.2. *Uma OWHF satisfaz as seguintes condições:*

- *A descrição de H deve ser publicamente conhecida, com nenhuma informação secreta para seu funcionamento.*
- *O argumento s pode ter comprimento arbitrário, e o resultado $H(s)$ tem um comprimento fixo de n bits (com $n \geq 64$).*
- *Dado H e s, o cálculo de $H(s)$ deve ser "fácil".*
- *A função de hash deve ser unidirecional no sentido de que, dado um Y na imagem de H, é "difícil" encontrar uma mensagem s tal que $H(s) = Y$, e dado s e $H(s)$, é "difícil" encontrar uma mensagem $s' \neq s$ tal que $H(s') = H(s)$.*

3.1.2.2 Funções de Hash Resistentes a Colisões (C*ollision Resistant Hash Functions* - CRHFs)

A definição a seguir é uma extensão da 3.1.2, de modo que o termo "função de hash unidirecional forte" também seria aplicável a CRHFs. No entanto, CRHF é preferível, pois explica mais claramente a propriedade real que é satisfeita [19].

Definição 3.1.3. *Uma CRHF satisfaz as seguintes condições:*

- *A descrição de H deve ser publicamente conhecida, com nenhuma informação secreta para seu funcionamento.*
- *O argumento s pode ter comprimento arbitrário, e o resultado $H(s)$ tem um comprimento fixo de n bits (com $n \geq 128$).*
- *Dado H e s, o cálculo de $H(s)$ deve ser "fácil".*
- *A função de hash deve ser unidirecional no sentido de que, dado um Y na imagem de H, é "difícil" encontrar uma mensagem s tal que $H(s) = Y$, e dado s e $H(s)$, é "difícil" encontrar uma mensagem $s_0 \neq s$ tal que $H(s_0) = H(s)$.*
- *A função de hash deve ser resistente a colisões: isso significa que é "difícil" encontrar duas mensagens distintas que resultem no mesmo valor de hash.*

3.2 Funções Aprovadas pelo NIST

O National Institute of Standards and Technology (NIST) é amplamente reconhecido como a principal entidade responsável por avaliar as famílias de funções de hash criptográficas nos Estados Unidos, servindo como referência para muitos outros países.

Os *Secure Hash Algorithms* (SHA) são algoritmos considerados seguros pelo NIST e fazem parte da família de funções de hash aprovadas, ou seja, consideradas seguras para aplicações práticas. O NIST disponibiliza informações detalhadas sobre esses algoritmos, incluindo análises de segurança e políticas acerca de seu uso, atualizadas à medida que surgem novos problemas que possam comprometer a segurança das versões existentes. Entre os padrões publicados, destacam-se o FIPS 180-1, que define o *Secure Hash Standard* (SHS) e descreve o SHA-1; o FIPS 180-4, que formaliza o SHA-2; e o FIPS 202, que especifica o SHA-3.

Na Figura 3.2, extraída do site do NIST [5], estão apresentadas as medidas de resistência em bits para as funções de hash que consistem no SHA. Essa resistência quantifica o número de tentativas que um adversário precisaria realizar para quebrar os principais problemas de segurança, que são representados na tabela na seguinte ordem: colisão, pré-imagem e segunda pré-imagem.

Por exemplo, uma resistência de 128 bits indica que um atacante precisaria, em média, realizar 2^{128} tentativas para encontrar uma colisão.

O valor de $L(M)$ é definido pelo NIST como:

$$L(M) = \log_2 \frac{\text{len}(M)}{B}$$

em que $\text{len}(M)$ representa o comprimento da mensagem M em bits e B é o tamanho do valor do hash gerado, também medido em bits.

	Collision Resistance Strength in bits	Preimage Resistance Strength in bits	Second Preimage Resistance Strength in bits
SHA-1	<80	160	160 – L (M)
SHA-224	112	224	min(224, 256 – L (M))
SHA-256	128	256	256 – L (M)
SHA-384	192	384	384
SHA-512	256	512	512 – L (M)
SHA-512/224	112	224	224
SHA-512/256	128	256	256
SHA3-224	112	224	224
SHA3-256	128	256	256
SHA3-384	192	384	384
SHA3-512	256	512	512

Figura 3.2 Medidas de resistência para as funções do padrão SHA [5].

Há uma diferença conceitual entre as funções SHA-1 e SHA-2 em relação ao SHA-3. Enquanto as duas primeiras baseiam-se em um esquema de construção comum, o SHA-3 utiliza uma abordagem diferente. Um panorama desses algoritmos será apresentado a seguir.

3.2.1 Funções de Compressão

Os algoritmos de funções hash criptográficas são frequentemente baseados em um caso específico da definição geral de hash, conhecido como função de compressão. Portanto, é importante primeiro abordar essa definição.

Essas funções têm o domínio restrito a entradas de tamanho fixo, ou seja, comprimem entradas de m bits para saídas de n bits, sendo $m > n$. A função de compressão é fundamental para a construção desses algoritmos, pois, análoga ao funcionamento de cifras de bloco, ela é responsável pela computação de partes de mensagens de tamanho arbitrário, dividindo-as em blocos de tamanho fixo. Essa abordagem facilita a manipulação de grandes volumes de dados e permite a introdução de lógicas adicionais de segurança nos algoritmos.

Definição 3.2.1. *Sejam n e m inteiros positivos, em que m é maior que n. Uma função de compressão f pode ser definida como:*

$$f : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$$

Nessa definição, os elementos do conjunto $\{0,1\}^m$ representam os blocos de dados de entrada, enquanto os elementos de $\{0,1\}^n$ são os resultados da compressão. A função f aceita um bloco de tamanho m e um valor de estado de comprimento n, produzindo um valor de saída também de comprimento n. Dessa forma, os valores de entrada podem ser vistos como uma combinação de dados e um estado, utilizados pela função de compressão para produzir um novo valor de hash.

Dado que o algoritmo de uma função de hash é projetado para aceitar entradas de tamanhos arbitrários, mas a função de compressão trata a mensagem por partes, uma técnica comum é a de preenchimento (*padding*), que consiste em garantir que a entrada tenha um comprimento apropriado para ser processada pela função de compressão. O preenchimento geralmente envolve incorporar bits adicionais à mensagem original, de forma que ela atinja um tamanho que seja um múltiplo do tamanho do bloco esperado pela função de compressão.

Em contextos em que o valor de hash será usado para alguma verificação, é comum que o preenchimento siga uma lógica específica para garantir maior confiabilidade no algoritmo. Assim, os bits não precisam ser transmitidos junto com a mensagem, desde que remetente e destinatário concordem com uma convenção para essa lógica [15].

3.2.2 Primeiras Funções Aprovadas

As funções das famílias anteriores ao SHA-3 utilizam o Esquema Merkle-Damgård em sua construção.

3.2.2.1 Esquema Merkle-Damgård

O esquema Merkle-Damgård é uma aplicação amplamente adotada de funções de hash iteradas, que consiste em processar a mensagem original em blocos menores de tamanho fixo (geralmente de 512 bits). Cada bloco é processado por uma função de compressão, responsável por gerar um valor de hash intermediário, e esses valores intermediários são combinados ao longo de várias iterações, até que o último valor seja produzido, resultando no hash final.

Durante o processamento, a função de compressão executa operações como deslocamentos e manipulações lógicas bit a bit, que podem variar conforme a implementação do algoritmo. Essas variações influenciam diretamente a segurança e a eficiência do cálculo.

Esse esquema expande o domínio originalmente previsto pela função de compressão utilizada, permitindo que ela lide com entradas de tamanho arbitrário. Além disso, se a função de compressão utilizada for resistente a colisões, a função de hash resultante também será [13].

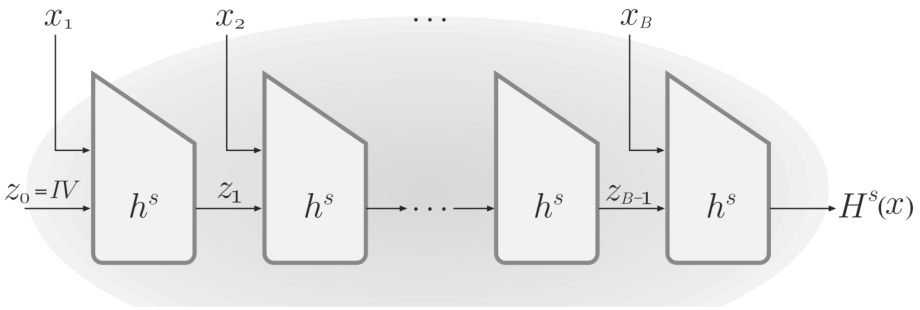

Figura 3.3 Diagrama do esquema Merkle-Damgård [13].

Na figura 3.3, o vetor de inicialização (IV, do inglês *initialization vector*) é um valor fixo e público, previamente estabelecido como padrão para o algoritmo, e também representa o primeiro estado Z_0. Os estados seguintes são representados por $Z_1, Z_2, \ldots, Z_{B-1}$. A mensagem original x é dividida em B blocos, representados por $x_1, x_2, \ldots, x_B$. A função de hash é representada pela notação H_s, e a função de compressão é indicada como h^s.

Cada bloco é combinado com o valor de estado gerado pela função de compressão do bloco anterior, até que todos os blocos sejam processados. O valor final da função de compressão do último bloco x_B é o hash final, denotado como $H_s(x)$. Esse método garante que todos as partes da mensagem contribuam para o hash.

Embora o esquema Merkle-Damgård seja amplamente utilizado em muitas funções de hash, ele possui vulnerabilidades conhecidas, como ataques de extensão de comprimento (*length extension attacks*). Esse tipo de ataque ocorre quando um adversário, ao conhecer o valor de hash de uma mensagem parcial, consegue forjar um novo hash válido sem precisar conhecer a mensagem original completa.

Nesse sentido, foram introduzidas algumas técnicas de fortalecimento do esquema, sendo uma das mais conhecidas o conceito de fortalecimento de Merkle-Damgård. Nesse fortalecimento, além do preenchimento comum, é incluída uma codificação do comprimento da mensagem original no final do último bloco, representada em bits e inserida após o preenchimento.

Esse mecanismo assegura que, mesmo que duas mensagens distintas compartilhem o mesmo prefixo — a sequência inicial (sem o preenchimento) de caracteres comum a ambas — elas resultarão em hashes completamente distintos. Caso um adversário tente modificar este prefixo, a alteração no comprimento será refletida, resultando em um hash diferente. Dessa forma, além de prevenir a criação de colisões, esse fortalecimento garante a integridade da mensagem.

3.2.2.2 O Precursor: MD

O MD5 (*Message Digest 5*) foi proposto por Rivest [21] em 1991 para aplicação em esquemas de assinaturas digitais. É uma evolução de sua versão anterior, o MD4, sendo ambos projetados com base no esquema Merkle-Damgård. O algoritmo utiliza uma função de compressão que processa blocos de 512 bits, resultando em um *digest* final de 128 bits.

Descrevendo seu funcionamento básico, a mensagem inicial é estendida com um padding que consiste em um único bit "1" seguido por bits "0" suficientes para que o comprimento total da mensagem seja congruente a 448 modulo 512. Após essa extensão, uma representação de 64 bits do comprimento original da mensagem é concatenada ao resultado anterior, assegurando que o comprimento total da mensagem resultante seja um múltiplo exato de 512 bits. Essa mensagem é então dividida em blocos de 512 bits, cada um contendo 16 palavras de 32 bits. O processamento de cada bloco ocorre em várias etapas, utilizando um registro de quatro palavras que é inicializado com um valor constante. O registro, neste sentido, se refere às variáveis que armazenam os resultados intermediários durante o processamento.

Embora o MD5 tenha sido rapidamente comprometido — uma vez que, em 1992, den Boer e Bosselaers encontraram colisões em sua função de compressão [19] — ele teve grande importância histórica, com seu processo de computação servindo como base para o SHA-1, por exemplo.

3.2.2.3 SHA-1

Desenvolvido pelo NIST em 1995, o SHA-1 surgiu como resposta à sua primeira versão, que tinha como objetivo ser uma alternativa mais robusta ao MD5 e apresentou vulnerabilidades rapidamente identificadas: o SHA-0 (1993). O SHA-1 produz hashes de 160 bits, em contraste com os 128 bits gerados pelo MD5.

Já em 2005, a publicação de Wang et al. [26] demonstrou que colisões no SHA-1 poderiam ser encontradas com uma complexidade menor que 2^{69} passos, alertando para as vulnerabilidades no algoritmo. No entanto, foi apenas em 2011 que o NIST o descontinuou oficialmente, e seu uso em assinaturas digitais foi proibido em 2013. Mesmo assim, o algoritmo continuou a ser empregado em diversas aplicações por um período considerável, especialmente devido a sistemas legados que necessitaram de tempo para se adaptar às novas orientações.

Em 2017, a Google publicou os resultados de uma pesquisa que apresentou a primeira técnica comprovada de geração de colisões no SHA-1, denominada "*SHAttered Attack*" [1]. A empresa já havia deixado de usar o SHA-1 anos antes, mas esse ataque reforçou a urgência de descontinuá-lo definitivamente e alertar a comunidade. A técnica demonstrou como gerar dois arquivos PDF distintos que produzem o mesmo hash usando o SHA-1, confirmando o ataque teórico publicado por Wang. Esse ataque foi mencionado em uma publicação do Git de 2023, que anunciou a transição do SHA-1 para o SHA-256 [3]. O Git também planeja abstrair sua base de código para facilitar futuras migrações para outras funções hash, caso necessário.

Ainda recentemente, em dezembro de 2022, o NIST anunciou um plano para reduzir ainda mais a dependência do SHA-1, destacando a necessidade de migração para alternativas mais seguras.

3.2.2.4 SHA-2

Embora o SHA-1 tenha continuado a ser utilizado por muitos anos, a família de algoritmos SHA-2 foi anunciada em 2002 pelo NIST no FIPS 180-4 e inclui funções de hash de maior comprimento: SHA-224, SHA-256, SHA-384, SHA-512 e suas variações. O número ao lado indica o tamanho do valor de hash gerado; no caso do SHA-224 e do SHA-384, eles são resultados do SHA-256 e do SHA-512, respectivamente, mas com o tamanho truncado.

Essas funções seguem uma construção muito parecida com a do SHA-1, ou seja, partem do esquema Merkle-Damgård. No entanto, diferenciam-se pelos tamanhos utilizados nas etapas do processo iterativo. Apesar de altamente recomendadas, especialmente como alternativa ao SHA-1, existem trabalhos que expõem vulnerabilidades em ataques direcionados a formas simplificadas do SHA-2, como levantado por Olivier [17].

Nesse sentido, em 2007, o NIST publicou um edital no Federal Register [2], anunciando uma competição aberta com o objetivo de encontrar um substituto para o SHA-2. Essa iniciativa foi uma precaução do instituto, considerando que a família SHA-2 poderia se tornar vulnerável a ataques a qualquer momento, especialmente por ter a mesma construção de algoritmos que já foram quebrados.

3.2.3 Novo Padrão: SHA-3

A competição para definir o novo padrão SHA-3 recebeu 64 submissões, das quais 51 foram selecionadas para a primeira fase de avaliação. Em julho de 2009, 14 algoritmos avançaram para a segunda fase, mas até o final de setembro daquele ano, alguns deles já haviam sido comprometidos [20].

Em 2 de outubro de 2012, o NIST selecionou o algoritmo Keccak como o padrão SHA-3, com base nas avaliações e revisões realizadas durante a competição. Esta proposta vencedora é baseada em uma variação da chamada "Construção Esponja". A equipe responsável pelo Keccak — composta por Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche e Ronny Van Keer — disponibiliza informações detalhadas sobre o algoritmo em seu site [4].

Comparado ao esquema Merkle-Damgård, a construção esponja oferece uma abordagem mais flexível, permitindo a definição do tamanho de saída de forma independente. Enquanto no Merkle-Damgård cada bloco de entrada é combinado linearmente com o estado anterior, o que pode levar a vulnerabilidades, como ataques de extensão, a construção esponja absorve a entrada de forma não sequencial. Utilizando operações de permutação, ela garante uma mistura mais robusta e segura dos dados, resultando em uma estrutura que não depende da saída anterior, aumentando a segurança e a integridade do sistema.

3.2.3.1 Construção Esponja

Segundo a definição da equipe Keccak, a construção esponja é uma generalização tanto das funções de hash, que têm saída de comprimento fixo, quanto das cifras de fluxo, que operam com entradas de comprimento fixo. Essa construção permite que uma função, chamada de função esponja, aceite uma cadeia binária de qualquer comprimento como entrada e retorne

uma cadeia binária de qualquer comprimento definido [4].

Como ilustrado na Figura 3.4, a construção esponja possui duas fases, nomeadas de forma análoga ao uso de uma esponja: absorção (*absorbing*) e compressão (*squeezing*). Ela consiste em três componentes principais: o estado de memória, dividido em duas partes, sendo r, conhecido como taxa (*rate*), que corresponde à quantidade de bits processados por bloco, e c, chamado de capacidade (*capacity*); a função de compressão, representada por f na imagem; e a função de preenchimento (*padding*), denotada por *pad*. A construção funciona de maneira iterativa, baseada neste estado de memória de tamanho fixo, com a função de compressão aplicando operações de permutação que garantem a difusão e a integridade dos dados.

Na inicialização, a mensagem M é concatenada com o preenchimento fornecido pela função *pad*, permitindo que seja separada em blocos de tamanho r bits. Para ser compatível com a construção esponja, *pad* não pode resultar em uma cadeia vazia e deve garantir que mensagens distintas, mesmo com tamanhos ou padrões similares, tenham preenchimentos diferentes, evitando ambiguidade [17]. Todos os bits do estado (ou seja, tanto os primeiros r quanto os c) são inicializados com zeros.

Na fase de absorção, os blocos de entrada de r bits são aplicados ao estado por meio de uma operação XOR (OU exclusivo), intercalados com aplicações da função f. Após processar todos os blocos de entrada, a construção avança para a fase de compressão.

A fase de compressão é responsável por determinar a saída da construção, com base no estado obtido da fase anterior e no tamanho desejado para a saída. Nesta fase, os primeiros r bits do estado são retornados como blocos de saída de forma iterativa, onde cada bloco de saída é intercalado com aplicações da função de compressão f. É importante ressaltar que, durante essa fase, os últimos c bits do estado permanecem inalterados e não são utilizados na geração da saída. Esses bits não são diretamente afetados pelos blocos de entrada e, portanto, não são retornados pela função. Essa característica assegura que uma parte do estado permaneça reservada, contribuindo para a segurança geral da construção.

Para a definição do SHA-3, os tamanhos de saída foram fixados em 224, 256, 384 e 512 bits, resultando em quatro algoritmos na família SHA-3: SHA3-224, SHA3-256, SHA3-384 e SHA3-512.

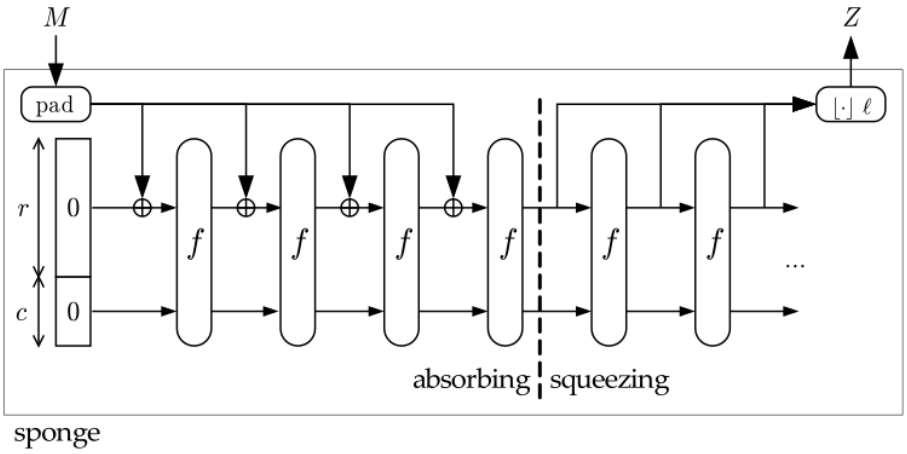

Figura 3.4 Diagrama da construção Esponja [4].

CAPÍTULO 4

Aplicações

As funções de hash são projetadas para atender a diversos objetivos relacionados à compressão de informações. Por exemplo, em operações internas de estruturas de dados, como tabelas de hash ou dicionários em linguagens de programação, essas funções devem ser computacionalmente rápidas, garantindo eficiência em operações como busca e inserção. Outro exemplo é a deduplicação de dados, que utiliza a impressão digital (*fingerprint*) de um arquivo para evitar duplicatas em servidores, economizando espaço de armazenamento e melhorando a eficiência de backup e recuperação.

Na perspectiva da Segurança da Informação, área escolhida para validar as aplicações das funções de hash neste trabalho, o foco recai sobre questões de integridade e autenticidade. A integridade assegura que os dados não foram alterados, corrompidos ou manipulados durante a transmissão ou armazenamento, enquanto a autenticidade refere-se à verificação da origem dos dados, garantindo que o remetente é realmente quem afirma ser.

O funcionamento dessas funções é crucial para a detecção de alterações nos dados, desempenhando um papel mais significativo na identificação do que na prevenção de modificações [24]. Em outras palavras, se um adversário modificar uma mensagem cifrada antes de seu recebimento, as funções de hash podem identificar essa alteração, garantindo a integridade das informações.

Historicamente, diversas tentativas foram feitas para abordar essas questões. Conforme observado por Stinson e Paterson [24], esquemas criptográficos tradicionais, embora eficazes contra adversários que interceptam mensagens, nem sempre oferecem garantias suficientes de segurança. As funções de hash permitem abordagens mais robustas, sendo consideradas os pilares[1] da criptografia moderna. Elas são amplamente reconhecidas como uma das principais primitivas criptográficas, atuando como uma representação única de uma mensagem ao gerar um valor (ou "imagem") que a identifica, comumente chamado de *digest* ou impressão digital (*digital fingerprint*) [15].

Além disso, frequentemente atuam como pré-processadores dos dados originais, eliminando a necessidade de projetar outras primitivas para entradas de tamanhos variados. Em assinaturas digitais, por exemplo, o computação de hashs torna o processo significativamente mais eficiente em comparação com a assinatura direta de mensagens inteiras. Assinar uma mensagem completa exigiria dividi-la em blocos e assinar cada um, resultando em uma assinatura final que cresce proporcionalmente ao tamanho da mensagem, tornando o processo ineficiente em termos de tempo e espaço.

Nas seções a seguir, serão apresentadas essa e outras aplicações relevantes na área. Em destaque, as duas primeiras tratam dos códigos de autenticação de mensagem (MACs) e das as-

[1] Termo interpretado do inglês "*workhorses*", utilizado por Schneier [22].

sinaturas digitais, que estão relacionadas a outras primitivas criptográficas que, assim como as funções de hash, desempenham um papel fundamental na segurança da informação. Os MACs utilizam criptografia simétrica para autenticação, enquanto as assinaturas digitais garantem a autenticidade e integridade da mensagem por meio de criptografia assimétrica. Por isso, as funções de hash podem ser vistas como uma ponte entre criptografia simétrica e assimétrica (ver Apêndice A), devido às suas aplicações em ambas as configurações [13].

A partir deste ponto, o termo "funções de hash" passa a se referir especificamente a "funções de hash criptográficas", uma vez que as aplicações mencionadas requerem suas garantias mais rigorosas de segurança, conforme discutido no Capítulo 3.

4.1 Códigos de Autenticação de Mensagem

Embora o termo MAC se refira à classe de funções descrita anteriormente na definição 3.1.1, ele também é utilizado para descrever uma primitiva criptográfica fundamental e amplamente empregada para garantir a integridade e autenticidade das mensagens em diversos sistemas de segurança.

No contexto da criptografia, o Código de Autenticação de Mensagem (*Message Authentication Code* - MAC), também conhecido como TAG, é um código gerado a partir da combinação da mensagem original com uma chave secreta compartilhada entre o remetente e o destinatário. No lado do remetente, o MAC é gerado com base na mensagem e na chave, sendo transmitido junto com a mensagem. No lado do destinatário, o MAC é recalculado com base na mensagem recebida e na chave secreta, permitindo verificar se o valor gerado corresponde ao MAC recebido. Caso os valores sejam idênticos, isso indica que a mensagem não foi alterada e que ela realmente provém do remetente legítimo.

MACs são essenciais para garantir a integridade e autenticidade das mensagens, prevenindo ataques como a falsificação ou modificação de mensagens durante a transmissão. Protocolos como o *Transport Layer Security* (TLS) e o *Secure Socket Layer* (SSL) utilizam MACs em seus mecanismos de segurança, protegendo dados que trafegam na internet contra interceptações e modificações. Por exemplo, no processo de *handshake* e na transmissão de dados no TLS, um MAC é gerado e anexado às mensagens para permitir que o receptor valide sua integridade e autenticidade. Se qualquer parte da mensagem for modificada durante o envio, o valor do MAC gerado no lado do destinatário não coincidirá com o MAC original, alertando sobre a violação da mensagem.

4.1.1 Hash e MAC (Hash-and-MAC)

Uma abordagem comum para aumentar a segurança de um MAC é combinar uma função de hash criptográfica resistente a colisões (*collision-resistant hash function* - CRHF) com o processo de geração do MAC, resultando no que é conhecido como *Hash-and-MAC*. Nessa abordagem, a mensagem é primeiro compactada por uma função de hash e, em seguida, essa saída é utilizada na geração do MAC. Isso permite que o processo de autenticação seja mais eficiente, especialmente para mensagens longas, reduzindo a quantidade de dados que precisam ser processados pelo MAC diretamente.

Essa combinação aumenta a segurança, pois, se um adversário interceptar a mensagem e o MAC, ele não poderá gerar um novo MAC válido sem a chave secreta, a qual não tem acesso. Caso tente modificar a mensagem original, a função de hash irá gerar um valor completamente diferente devido ao efeito avalanche, que assegura que mudanças mínimas na entrada resultem em grandes alterações no valor de hash. Além disso, a resistência a colisões da função de hash torna extremamente difícil encontrar outra mensagem que gere o mesmo valor de hash usado para o MAC, prevenindo ataques de falsificação.

Um exemplo amplamente utilizado dessa estratégia é o Código de Autenticação de Mensagem Baseado em Hash (*Hash-based Message Authentication Code* - HMAC). O HMAC, padronizado pelo NIST em 2005 com o FIPS 198-1, utiliza uma função de hash (como *SHA-256*) juntamente com uma chave secreta para gerar um MAC. O HMAC adiciona robustez ao incluir a chave no processo de hashing, combinando a mensagem e a chave de maneira que qualquer modificação na mensagem ou na chave resulte em uma saída de MAC completamente diferente. O HMAC é amplamente adotado em protocolos de segurança, como TLS e IPsec.

4.1.2 Outras Construções de MAC

Além do HMAC, outras construções de MAC também podem ser baseadas em funções de hash. Um exemplo é o MAC aninhado (*Nested* MAC), descrito por Stinson e Paterson [24], que constrói o MAC com base na composição de duas famílias de funções de hash baseadas em chave.

A construção de MACs baseados em funções de hash resultou na necessidade de que essas funções também possam ser usadas para criar funções pseudo-aleatórias (*pseudo-random functions* - PRFs), conforme observado por Preneel [20]. Uma PRF se diferencia de uma função de hash comum porque, ao ser chaveada, ela gera saídas que parecem aleatórias, mesmo sendo determinísticas para a mesma entrada e chave. Esse comportamento pseudo-aleatório é crucial em várias aplicações criptográficas, já que torna difícil para um adversário prever ou manipular a saída sem conhecer a chave secreta.

4.2 Assinaturas Digitais

Assinaturas manuscritas, como as feitas em papel, são amplamente utilizadas para validar a autenticidade de documentos. De maneira semelhante, um esquema de assinatura digital corresponde a uma maneira de autenticar uma mensagem armazenada em formato eletrônico. Essa definição é apresentada por Stinson e Paterson [24], que também destacam duas diferenças essenciais entre assinaturas convencionais e digitais.

A primeira diferença é que, ao contrário das assinaturas manuscritas, que estão diretamente vinculadas ao documento, a assinatura digital não possui essa ligação, no sentido de fazer parte do documento, ou da mensagem. A segunda diferença diz respeito ao processo de verificação. Enquanto uma assinatura convencional é verificada comparando-se a escrita ou o formato da assinatura com uma versão autenticada ou algum documento oficial, as assinaturas digitais são confirmadas usando informações públicas, o que significa que qualquer pessoa pode verificar a autenticidade de uma assinatura digital.

Entre os diversos usos das assinaturas digitais, destacam-se a validação de documentos em formatos eletrônicos, a utilização de certificados digitais para autenticação de usuários e servidores em protocolos de comunicação na internet, como TLS/HTTPS, e a autenticação de e-mails, que garante que a mensagem foi enviada pelo remetente declarado, assegurando a integridade e a autenticidade da comunicação.

Um esquema de assinatura digital envolve duas partes principais: o assinante (remetente) e o verificador (destinatário), sendo um exemplo clássico do uso da criptografia de chave pública. Inicialmente, o processo consistia em o assinante encriptar diretamente o documento ou mensagem utilizando sua chave privada, enquanto o verificador utilizava a chave pública do assinante para decriptar o conteúdo e verificar se ele correspondia ao documento original. Essa abordagem, embora funcional, logo se revelou ineficiente para mensagens de tamanho variável, especialmente para documentos mais longos, uma vez que o processo de encriptação direta consumia uma quantidade significativa de recursos computacionais.

O algoritmo RSA (Rivest-Shamir-Adleman), introduzido em 1977, foi amplamente adotado para esse tipo de criptografia, sendo baseado em um problema matemático considerado complexo: extração de uma raiz módulo um inteiro composto. Na prática, o RSA utiliza um par de chaves, composto pela chave pública (n, e), sendo n o módulo e e o expoente de encriptação. No entanto, o RSA apresenta algumas limitações, uma vez que ele só pode encriptar diretamente mensagens cujo tamanho seja menor ou igual ao valor de n, ou seja, ele não é adequado para encriptar documentos longos diretamente. Por conta disso, esse método inicial foi progressivamente substituído pela abordagem que utiliza funções de hash que, ao serem combinadas com o RSA, tornaram o processo de assinatura digital mais ágil e prático.

4.2.1 Hash e Assinaturas Digitais

Desde 1976, Diffie e Hellman [12] ressaltaram a importância das funções unidirecionais para garantir a segurança na criptografia de chave pública, o que abriu caminho para o desenvolvimento de assinaturas digitais mais eficientes. Além de reduzir o tamanho da entrada que precisa ser assinada, Damgård também destaca que o uso de uma função de hash ajuda a evitar ataques, como a inserção ou remoção de partes da mensagem [9].

Essas funções de hash geram o hash inicial e esse valor, que serve como a representação compacta do documento original, é o que será encriptado com a chave privada do assinante, resultando na criação da assinatura digital. Por sua vez, o verificador, ao receber o documento, gera o hash correspondente e decripta a assinatura utilizando a chave pública do assinante. Se o hash decriptado for idêntico ao hash gerado localmente, o verificador pode concluir que o documento é autêntico e não foi alterado.

Para assinaturas envolvendo múltiplos arquivos, Ralph Merkle propôs, em 1979, o uso de árvores de hash, em que cada folha representa o hash de um arquivo. A publicação do valor da raiz permite que os usuários verifiquem a integridade de um conjunto de arquivos, assegurando que sua versão local esteja em conformidade com o conjunto original. A diferença no valor da raiz indica a presença de um arquivo corrompido, e a publicação de outros hashes ajuda a restringir a busca pelo arquivo que precisa ser substituído. Para mais detalhes sobre o tema, recomenda-se a leitura de Page (2009) [18].

4.3 Outras Aplicações

1. **Verificação de integridade de software**: Existem aplicações diretas das funções de hash no contexto do desenvolvimento de software para garantir a integridade e identificação de arquivos.

   Em sistemas de controle de versão, como o Git, um exemplo básico é o hash de *commits* (termo utilizado para a entidade que corresponde a um determinado registro de mudanças no software), que serve como identificador das alterações que carrega. Para incorporar as mudanças associadas a um commit, seu hash atua como seu identificador — e, de fato, o valor obtido é derivado das modificações que ocorreram no código, além de seus metadados, como o autor. Isso significa que qualquer alteração nos arquivos resultará em um novo hash, permitindo que desenvolvedores e usuários detectem rapidamente se o conteúdo foi alterado de forma não autorizada.

   No contexto da distribuição de software, uma imagem ISO de sistemas operacionais é um arquivo que contém todos os dados necessários para a instalação do sistema. Para garantir que essa ISO não foi corrompida durante o download, um hash é gerado a partir dela e disponibilizado publicamente pelo distribuidor. Dessa maneira, os usuários podem calcular novamente o hash localmente e compará-lo ao hash publicado, verificando se sua versão local é válida ou não, sendo importante para verificar versões corrompidas ou maliciosas.

2. **Autenticação baseada em senhas**: A maioria dos sistemas digitais adota um modelo de autenticação em que cada usuário — representando uma pessoa ou entidade — está vinculado a uma senha. Para aumentar a segurança, em vez de armazenar a senha original, os sistemas mantêm apenas seu valor hash. Ao inserir a senha, o sistema gera o hash correspondente e o compara ao valor armazenado. A segurança do sistema depende da robustez da função hash utilizada, implicando que, mesmo que os hashes sejam vazados, os valores originais permanecem, em teoria, protegidos. Contudo, na prática, esse método enfrenta desafios, pois as senhas costumam seguir padrões previsíveis, devido a serem frequentemente escolhidas por seres humanos. Adversários podem explorar fraquezas (como os padrões) para conduzir ataques eficazes, como os ataques de dicionário.

   Diante dessas vulnerabilidades, é essencial considerar técnicas que aprimorem a segurança associada ao uso de funções de hash em senhas, como discutido em trabalhos anteriores, incluindo [10] e [25]. Em especial, destaco a utilização da técnica de *salting*, que consiste em adicionar uma cadeia de caracteres aleatória (o "sal") à entrada antes da aplicação da função de hash; essa abordagem garante um nível adicional de aleatoriedade na computação do valor de hash, assegurando que duas entradas idênticas não gerem o mesmo hash.

   Além disso, iniciativas como a competição para o SHA-3 inspiraram a busca por melhores práticas específicas na proteção de senhas, incluindo a *Password Hashing Competition*, anunciada em 2013. Essa competição resultou na escolha da família de esquemas para hash de senhas Argon2 como padrão recomendado [6].

3. **Derivação de Chaves**: Este conceito é fundamental na criptografia e refere-se à geração de chaves criptográficas a partir de uma chave original ou mestre.

   A derivação de chaves evita o reuso de uma única chave para diferentes propósitos, garantindo que apenas a chave original precise ser armazenada, e as funções de hash podem atuar como uma ferramenta de derivação de chaves, sendo conhecidas pelo termo extratores fortes (*strong extractors*) [24]. Essas novas chaves são seguras e aleatórias, desde que a chave inicial não siga padrões previsíveis.

   O processo de derivação de chaves é especialmente importante em ambientes que requerem segurança temporária, como na geração de senhas de uso único (*one-time passwords* - OTPs) [15], essenciais para validar uma única sessão ou atuar como parte de um sistema de autenticação de dois fatores. Protocolos de segurança, como o WPA (*Wi-Fi Protected Access*), utilizam funções de hash durante a autenticação para garantir que cada conexão tenha uma chave única, promovendo uma comunicação mais segura entre os dispositivos. Da mesma forma, o protocolo Kerberos, amplamente utilizado em redes corporativas, depende de funções de hash; ao se autenticar, o usuário recebe um tíquete criptografado que inclui uma chave de sessão derivada, garantindo que a comunicação entre o usuário e os serviços seja segura e que apenas usuários autorizados possam acessar os recursos da rede.

CAPÍTULO 5

Conclusão

As funções de hash, exploradas neste trabalho, constituem um conceito amplo e complexo. Autores como Preneel [19] propõem iniciativas para minimizar o efeito de "Babel de Línguas" (*Babel of Tongues*) — uma analogia à Torre de Babel — com o objetivo de simplificar o entendimento na área. Nesse sentido, é notável a sobrecarga de termos que se referem ao valor obtido pelas funções de hash, variando conforme o contexto em que são utilizadas.

Elas podem ser classificadas de maneira mais simplificada em funções baseadas em chave e não baseadas em chave; bem como de forma mais específica, de acordo com suas aplicações, dividindo-se entre Funções de Códigos de Autenticação de Mensagens (MAC) e Funções de Códigos de Detecção de Modificação (MDC).

Ainda, conforme discutido, a segurança é um estado dinâmico e momentâneo; com o avanço do poder computacional, a eficácia de algoritmos anteriormente considerados seguros pode ser colocada em prova, o que ressalta a necessidade de estudos contínuos e de revisões regulares nas práticas de segurança. Embora as funções de hash representem apenas uma parte dessa compreensão crítica, é essencial que desenvolvedores se familiarizem com as características dos algoritmos que utilizam.

Na análise das funções de hash, os principais requisitos de segurança envolvem a resistência aos problemas de Pré-imagem, Segunda Pré-imagem e Colisão, que são utilizados para medir a robustez dos algoritmos. Com o tempo, o aumento das demandas de segurança, impulsionado por ataques bem-sucedidos e pela necessidade de maior aderência a esses requisitos, levou ao desenvolvimento de algoritmos consolidados, como os aprovados pelo *National Institute of Standards and Technology* (NIST). Atualmente, as famílias SHA-2 e SHA-3 são padrões amplamente reconhecidos e recomendados.

Este trabalho também identificou tópicos que não foram abordados em profundidade, dada a necessidade de se adequar ao tempo e escopo previstos para a elaboração de seu texto. Durante a pesquisa, surgiram temas de interesse que merecem exploração futura. Um desses temas é a implementação mais aprofundada das funções de hash criptográficas, a fim de compreender quais são as dificuldades associadas às operações e problemas matemáticos que sustentam sua resistência. Além disso, a competição pelo padrão SHA revelou outras funções, além do Keccak, que poderiam ser investigadas em estudos futuros.

Por fim, observou-se uma intersecção significativa entre os temas de segurança e criptografia. Embora a criptografia não tenha sido o foco principal deste estudo, sua presença em diversos esquemas de segurança é inevitável, evidenciando a importância de um entendimento integrado dessas disciplinas para a construção de soluções robustas em segurança da informação.

APÊNDICE A

Notas em Criptografia

A criptografia é o estudo de métodos para codificar mensagens, e suas raízes remontam ao tempo de Júlio César (100 a.C.). Seu propósito fundamental é ocultar o significado ou a intenção original de qualquer informação. Em sua forma mais simples, podemos visualizar um esquema criptográfico através das figuras de Alice e Bob, comumente utilizadas na literatura da área. Imagine que Alice deseja enviar uma mensagem secreta a Bob, mas a comunicação pode ser monitorada por um adversário.

Tradicionalmente, utiliza-se uma única chave secreta para codificar e decodificar a mensagem, prática conhecida como **criptografia simétrica**. Em contrapartida, a **criptografia assimétrica** emprega um par de chaves: uma para codificar e outra para decodificar a mensagem, sendo uma delas pública.

Na era da criptografia moderna, a noção de que o adversário possui recursos computacionais infinitos foi deixada de lado; hoje reconhecemos que esses recursos são limitados [11]. Assim, assume-se que o adversário é um algoritmo probabilístico que opera em tempo polinomial, assim como os algoritmos utilizados, que também são probabilísticos e funcionam em tempo polinomial, o que torna a compreensão de segurança mais realista e mensurável.

Durante este trabalho, alguns termos da criptografia foram adotados:

- **Primitiva Criptográfica**: Refere-se a um elemento essencial para esquemas criptográficos, incluindo, dentre outros, funções unidirecionais, cifras, funções de hash e códigos de autenticação de mensagem (MACs).
- **Chave**: Cadeia de bits utilizada nos processos de criptografia, que pode ser secreta ou pública, dependendo do esquema adotado.
- **Mensagem**: Conjunto de dados que precisa ser protegido ou autenticado, abrangendo texto, arquivos, *software* ou documentos. O termo refere-se ao texto não criptografado, chamado também de texto plano (*plain text*), enquanto a mensagem criptografada é conhecida como cifrotexto (*ciphertext*). O uso de "mensagens" neste trabalho para ilustrar a entrada da função de hash se baseia nesse conceito da criptografia.
- **Funções Unidirecionais**: De maneira geral, uma função unidirecional é aquela que é difícil de inverter. No contexto deste trabalho, o termo "unidirecional" se refere a uma propriedade de segurança. As funções de hash criptográficas possuem essa característica, o que as classifica como uma categoria específica de funções unidirecionais, mas existem outras funções unidirecionais que não são utilizadas como funções de hash.

Para uma compreensão mais aprofundada, recomendo a leitura de Katz e Lindell [13] e de Menezes et al. [15], que foram fundamentais para alguns conceitos relacionados às funções de hash discutidos. Além disso, para entender a evolução da área, o texto Novas Direções na Criptografia (*New Directions in Cryptography*), também mencionado neste trabalho, é uma leitura interessante.

Referências Bibliográficas

[1] Announcing the first SHA1 collision — security.googleblog.com. https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html. [Accessed 07-10-2024].

[2] Federal Register :: Request Access — federalregister.gov. https://www.federalregister.gov/d/E7-21581. [Accessed 30-09-2024].

[3] Git - hash-function-transition Documentation — git-scm.com. https://git-scm.com/docs/hash-function-transition. [Accessed 07-10-2024].

[4] Keccak — keccak.team. https://keccak.team/. [Accessed 07-10-2024].

[5] National Institute of Standards and Technology — nist.gov. https://www.nist.gov/. [Accessed 30-09-2024].

[6] Password Hashing Competition — password-hashing.net. https://www.password-hashing.net/. [Accessed 06-10-2024].

[7] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*, pages 1–15. Springer, 1996.

[8] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[9] Ivan Bjerre Damgård. Collision free hash functions and public key signature schemes. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 203–216. Springer, 1987.

[10] Lucas Grisi Oliveira de Queiroz. Funções de hash e gerenciamento de senhas. 2022.

[11] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. 2001.

[12] Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[13] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, 3rd edition, 2021.

[14] Hans Peter Luhn. Key word-in-context index for technical literature (kwic index). *American documentation*, 11(4):288–295, 1960.

[15] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Capítulo 9 - hash functions. In *Handbook of Applied Cryptography*, pages 321–383. CRC Press, 2001.

[16] Robert Morris. Scatter storage techniques. *Communications of the ACM*, 26(1):39–42, 1983.

[17] Gracielle Forechi Olivier. Estudo e implementação do algoritmo de resumo criptográfico sha-3. 2013.

[18] Thomas Page. *The application of hash chains and hash structures to cryptography*. PhD thesis, University of London, 2009. Chapter 3.

[19] Bart Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Citeseer, 2003.

[20] Bart Preneel. The first 30 years of cryptographic hash functions and the nist sha-3 competition. In *Cryptographers' track at the RSA conference*, pages 1–14. Springer, 2010.

[21] Ronald Rivest. Rfc1321: The md5 message-digest algorithm, 1992.

[22] Bruce Schneier. Cryptanalysis of md5 and sha: Time for a new standard. https://www.schneier.com/essays/archives/2004/08/cryptanalysis_of_md5.html, 2004. [Accessed 30-09-2024].

[23] H Stevens. Hans peter luhn and the birth of the hashing algorithm-ieee spectrum. *IEEE Spectrum: Technology, Engineering, and Science News*, 30, 2018.

[24] Douglas R. Stinson and Maura B. Paterson. *Cryptography: Theory and Practice*. CRC Press, 4th edition, 2019.

[25] Wayne C Summers and Edward Bosworth. Password policy: the good, the bad, and the ugly. In *Proceedings of the winter international symposium on Information and communication technologies*, pages 1–6, 2004.

[26] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *Advances in Cryptology–CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings 25*, pages 17–36. Springer, 2005.

[27] Arthur F Webster and Stafford E Tavares. On the design of s-boxes. In *Conference on the theory and application of cryptographic techniques*, pages 523–534. Springer, 1985.