

Segurança da Informação– GBC083

Prof. Rodrigo Sanches Miani – FACOM/UFU

Funções de hash

Prof. Rodrigo Sanches Miani – FACOM/UFU

Tópicos da aula

1. Ideia
2. Exemplo simples
3. Função de hash criptográfica
4. Exemplos
5. Aplicações



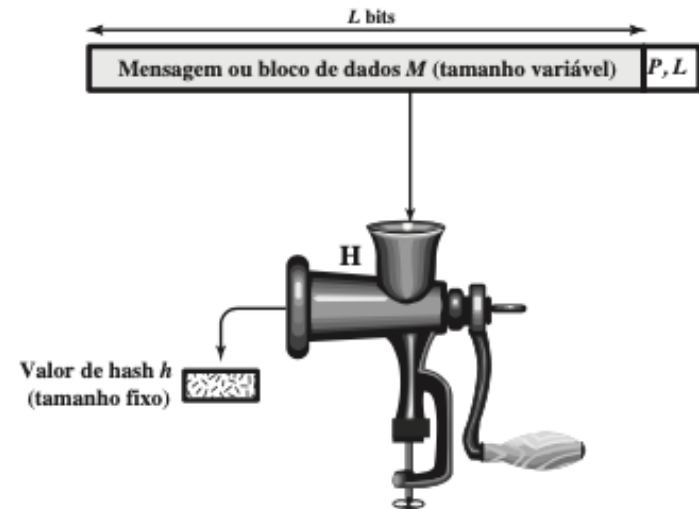
Ideia



Segurança da Informação– GBC083

Ideia

- ▶ Uma **função hash** H recebe como argumento uma mensagem M de qualquer tamanho e retorna um valor de hash h de tamanho fixo:
 - ▶ $h = H(M)$
 - ▶ h também é conhecido como resumo de M



P, L = preenchimento mais campo de tamanho

Função de hash

- ▶ Uma boa função hash produz saídas bem distribuídas e aparentemente aleatórias para um conjunto de entradas diverso;
- ▶ Uma mudança em qualquer bit ou bits em M resulta, com alta probabilidade, em uma mudança no código de hash;
- ▶ Costumam ser conhecidas como “*one-way functions*” ou funções de uma via.

Exemplo simples

Segurança da Informação– GBC083

Função de Hash simples

- ▶ A entrada (mensagem, arquivo...) é vista como uma sequência de blocos de n bits;
- ▶ A entrada é processada um bloco de cada vez, para produzir uma função de hash de n bits;
- ▶ Uma das funções de hash mais simples é o XOR bit a bit de cada bloco:

$$C_i = b_{i1} \oplus b_{i2} \oplus \dots \oplus b_{im}$$

onde

C_i = i -ésimo bit do código de hash, $1 \leq i \leq n$

m = número de blocos de n bits na entrada

b_{ij} = i -ésimo bit no j -ésimo bloco

\oplus = operação XOR



Função de hash criptográfica

Segurança da Informação– GBC083

Requisitos

- ▶ Uma função de **hash criptográfica forte** deve respeitar os seguintes requisitos:
 1. Aceitar mensagens de entrada de tamanho variável.
 2. Retornar valores de tamanho fixo.
 3. Deve ser eficiente em termos de complexidade computacional.

Requisitos

- ▶ Uma função **hash criptográfica forte** deve respeitar os seguintes requisitos:
 - 4. **Resistência à 1ª inversão (pré-imagem)**, ou seja, é computacionalmente inviável, dado y , encontrar x tal que $h(x) = y$;
 - 5. **Resistência à 2ª inversão (segunda pré-imagem)**, ou seja, é computacionalmente inviável, dado x_1 , encontrar $x_2 \neq x_1$ tal que $h(x_1) = h(x_2)$;
 - 6. **Resistência a colisões**, ou seja, é computacionalmente inviável, encontrar quaisquer x_1 e x_2 tais que $h(x_1) = h(x_2)$.

Exemplos

Segurança da Informação– GBC083

Exemplos

- ▶ As funções das famílias MD (Message digest) e SHA (Secure Hash) são bem famosas...
- ▶ MD5 é uma função problemática do ponto de vista de segurança e avisos sobre não usá-la em aplicações críticas são feitos desde 2011;
- ▶ No começo de 2017 pesquisadores encontraram falhas (colisões) no SHA1. O ataque foi implementado com o auxílio da infraestrutura computacional do Google;
- ▶ A recomendação atual é utilizar funções como SHA2 e SHA3.
- ▶ <https://arstechnica.com/information-technology/2017/02/at-deaths-door-for-years-widely-used-sha1-function-is-now-dead/>
- ▶ <https://www.computerworld.com/article/3173616/security/the-sha1-hash-function-is-now-completely-unsafe.html>



SHA

- ▶ A função hash mais utilizada nos últimos anos é o SHA (*Security Hash Algorithm*);
- ▶ Foi desenvolvida pelo NIST e padronizada no documento *Secure Hash Standard* (FIPS 180).;
- ▶ O SHA-0, a primeira versão, foi substituída pelo SHA-1 em 1995 quando foram descobertas suas primeiras vulnerabilidades.

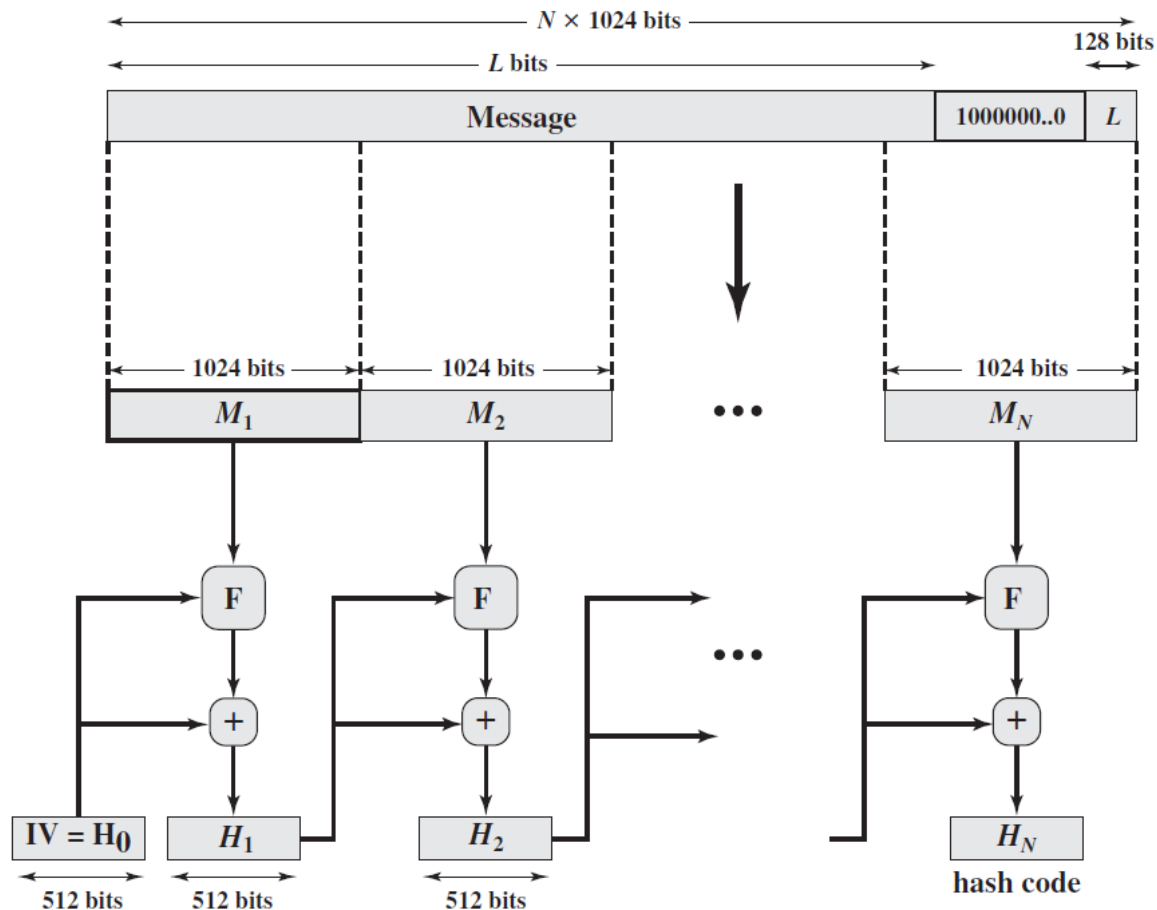
SHA

- ▶ O SHA-1 produz uma saída de 160 bits.
- ▶ Posteriormente, foi lançado o SHA-2, que produz saídas de diferentes tamanhos: 224 bits (SHA-224 e SHA-512/224), 256 bits (SHA-256 e SHA-512/256), 384 bits (SHA-384) e 512 bits (SHA-512).

SHA

- ▶ Em novembro de 2007 o NIST lançou um concurso para criar uma nova versão do SHA.
- ▶ Em dezembro de 2012 foi anunciado o algoritmo vencedor, batizado de SHA-3.
- ▶ O SHA-3 produz saídas nos seguintes tamanhos: 224 (SHA3-224) bits, 256 bits (SHA3-256), 384 bits (SHA3-384) e 512 bits (SHA3-512).
- ▶ <https://csrc.nist.gov/projects/hash-functions/sha-3-project>

SHA 512 – Visão geral



1. Adição de bits de padding e do tamanho do bloco;
2. Inicialização de um buffer que armazenará os resultados;
3. Processamento dos n blocos (1024 bits).;
4. A função f (função de compressão) produzirá 8 blocos de 64 bits - deslocamentos circulares, AND, OR, XOR;
5. A saída do estágio N é o hash da mensagem M .

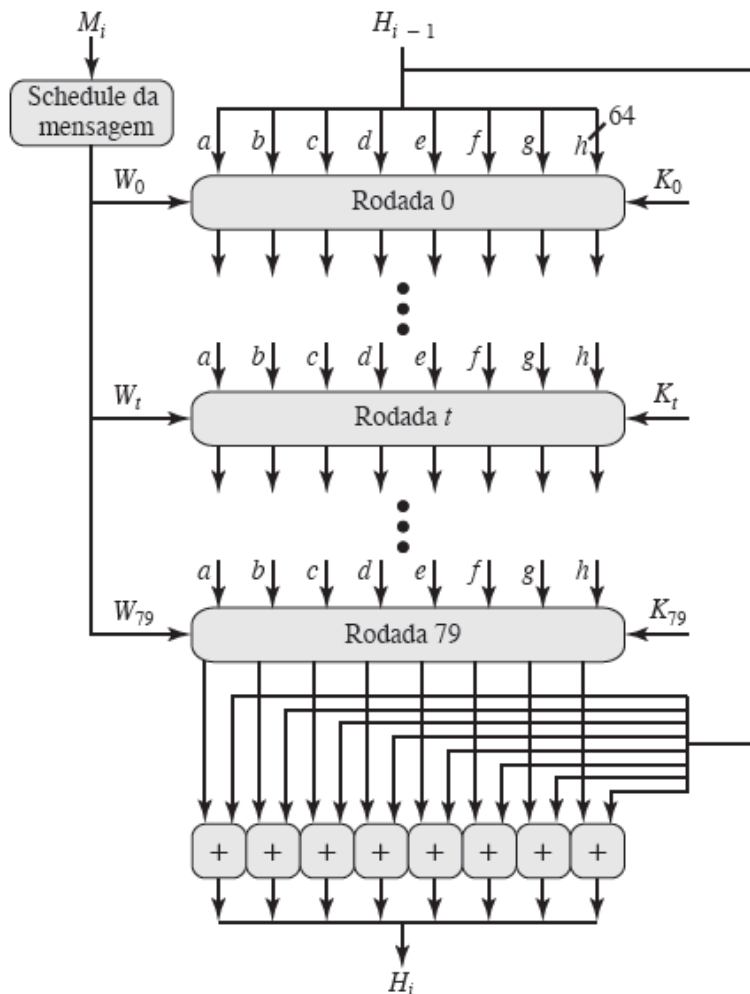
SHA 512 – Visão geral

- ▶ Passo 1: adição de bits de padding na mensagem para completar o tamanho de forma a atender o requisito $\text{tamanho} \equiv 896 \pmod{1024}$;
- ▶ Passo 2: um bloco de 128 bits representando o tamanho da mensagem é adicionado ao final da saída do passo 1.
 - ▶ Passo 1+ Passo 2 indica que a mensagem que servirá de entrada a função de hash possui um comprimento múltiplo de 1024 bits;
- ▶ Passo 3: inicialização de um buffer de 512 bits, que será o IV (vetor de inicialização) e irá guardar os resultados intermediários e finais do algoritmo.
 - ▶ 8 registradores de 64 bits (a, b, c, d, e, f, g, h)
 - ▶ Inicializados com os 64 primeiros bits das partes fracionárias das raízes quadradas dos oito primeiros números primos

SHA 512 – Visão geral

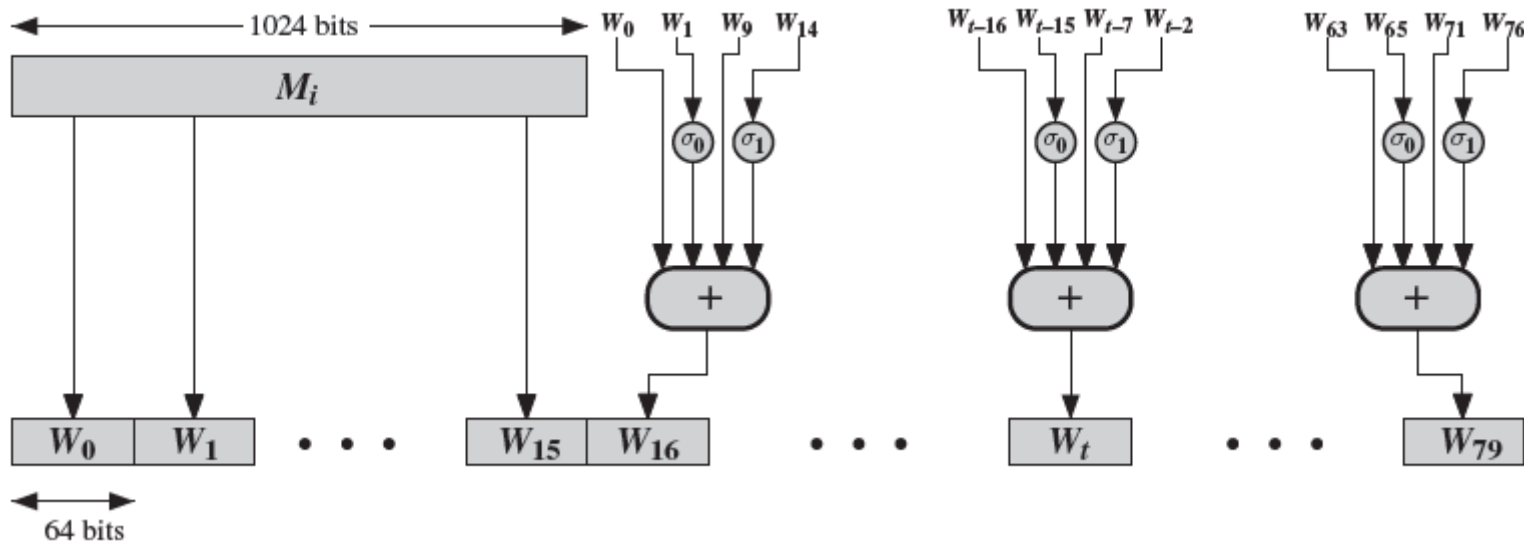
- ▶ Passo 4: mensagem é processada em blocos de 1024 bits. Cada bloco passa por 80 rodadas de processamento contidas em um módulo representado por F . A saída de um bloco é usada como uma das entradas do próximo.
 - ▶ Cada rodada faz uso de um valor de 64 bits derivado do bloco de 1024 que está sendo processado;
 - ▶ Cada rodada faz uso de uma constante de 64 bits (80 constantes) em que representam os primeiros 64 bits das partes fracionárias das raízes cúbicas dos primeiros 80 números primos.
 - ▶ Cada registrador de 64 bits (são 8!) passa por deslocamentos circulares, AND, OR, XOR usando os dois valores acima.
- ▶ Passo 5: a saída do processamento do último bloco é a saída do algoritmo.

SHA 512 – Visão geral



- ▶ $K_t = 80$ constantes (primeiros 64 bits das partes fracionárias das raízes cúbicas dos primeiros 80 números primos);
- ▶ M_i = i-ésimo bloco da mensagem original M ;
- ▶ W_t = t-ésimo palavra de 64 bits selecionada a partir do bloco original M_i ;
- ▶ A função *schedule da mensagem* é responsável por selecionar 64 bits do i-ésimo bloco da mensagem M ;
- ▶ Na prática, o que acontece são diversas operações entre $a, b, c, d, e, f, g, h, W_t, K_t$.
- ▶ Ao final da rodada 80, basta concatenar os valores de a, b, c, d, e, f, g, h para termos a saída de hash H_i .

SHA 512 – Visão geral



- ▶ As primeiras 16 palavras de 64 bits são selecionadas em ordem;
- ▶ As próximas 64 (80 rodadas – 16) são selecionadas a partir de uma combinação entre quatro palavras anteriores.

MD5 e SHA

```
[(base) MacBook-Pro-de-Rodrigo-7:~ rodrigomiani$ echo teste | md5
1ca308df6cdb0a8bf40d59be2a17eac1
[(base) MacBook-Pro-de-Rodrigo-7:~ rodrigomiani$ echo teste | shasum
9dc628289966d144c1a5fa20dd60b1ca1b9de6ed -
```

```
[(base) MacBook-Pro-de-Rodrigo-7:~ rodrigomiani$ echo testi | md5
e20752b108d3cb3dd47adb44e5535221
[(base) MacBook-Pro-de-Rodrigo-7:~ rodrigomiani$ echo testi | shasum
30b278045442d5501937a0df06eb81d58de839e9 -
```





Aplicações









Segurança da Informação– GBC083

Aplicações

1. Verificação de integridade de arquivo;
2. Armazenamento de senhas;
3. Autenticação de mensagem (MACs);
4. Assinatura digital.








Verificação de integridade de arquivo

	Name	Last modified	Size	Description
	Parent Directory		-	
	FOOTER.html	2020-08-06 12:30	810	
	HEADER.html	2020-08-07 06:01	3.9K	
	SHA256SUMS	2020-08-06 12:30	202	
	SHA256SUMS.gpg	2020-08-06 12:30	833	
	ubuntu-20.04.1-desktop-amd64.iso	2020-07-31 13:52	2.6G	



Verificação de integridade de arquivo

	Name	Last modified	Size	Description
	Parent Directory		-	
	FOOTER.html	2020-08-06 12:30	810	
	HEADER.html	2020-08-07 06:01	3.9K	
	SHA256SUMS	2020-08-06 12:30	202	
	SHA256SUMS.gpg	2020-08-06 12:30	833	
	ubuntu-20.04.1-desktop-amd64.iso	2020-07-31 13:52	2.6G	

b45165ed3cd437b9ffad02a2aad22a4ddc69162470e2622982889ce5826f6e3d *ubuntu-20.04.1-desktop-amd64.iso
443511f6bf12402c12503733059269a2e10dec602916c0a75263e5d990f6bb93 *ubuntu-20.04.1-live-server-amd64.iso

Armazenamento de senhas

- ▶ Ao criar um sistema de autenticação com usuário/senha, como a senha deve ser armazenada no banco de dados?
 - ▶ Guarda-se o hash da senha usando alguma função de hash criptográfica forte!
 - ▶ Quando o usuário for acessar o sistema, o sistema irá calcular o hash da senha fornecida e irá comparar com o que está armazenado no banco de dados. Se ambos forem iguais, a senha é a mesma e o acesso é liberado.
- ▶ Vantagens:
 - ▶ Terceiros não conseguem visualizar o conteúdo da senha;
 - ▶ Calcular *hash* é muito rápido!
- ▶ Atacantes podem usar um recurso chamado **Rainbow Table** para atacar senhas armazenadas dessa forma... Desenvolvedores podem usar um recurso chamado **salt** para se proteger dos ataques... Pesquisar esses recursos!



Armazenamento de senhas

- ▶ Existem protocolos (ferramentas) específicas para armazenar senhas em bancos de dados. Tais ferramentas são baseadas no procedimento visto anteriormente (hash+salt):
 - ▶ Bcrypt;
 - ▶ PBKDF2



Armazenamento de senhas – Rainbow tables

▶ Rainbow Tables:

- ▶ É uma tabela pré-calculada para encontrar o valor reverso de funções de hash, especialmente para quebrar hashes de senhas. Tais tabelas são usadas para encontrar senhas em claro, números de cartão de crédito e etc.
- ▶ A construção de rainbow tables envolve o mapeamento de uma determinada senha com o respectivo valor da função hash.
- ▶ Rainbow tables podem ser evitadas caso o sistema use um recurso conhecido como **salt** no momento do cálculo da função de hash.



Armazenamento de senhas – Rainbow tables

---TYPE	---HASH	---PASS	---STATUS	---TIME	---SUBMITTED
md5	7e89bcc6151b24992a255cd665d4aa16		waiting	0:0:46	2006-11-11 10:45:31
md5	0696eeaff05bf2105b0bcf6d93ac73a0		waiting	0:0:47	2006-11-11 10:45:30
md5	db549b9d18aabe8ad07aa3d9338d441c		waiting	0:1:38	2006-11-11 10:44:39
md5	70c9ecbd2512460fa861de25fb3d7c6e		waiting	0:24:8	2006-11-11 10:22:09
md5	c32cf089d464d3ed1a3af347ae208188		processing3	0:25:6	2006-11-11 10:21:11
md5	c6fe5851aff10a64e8a52e82b323304f		processing3	0:46:29	2006-11-11 09:59:48
md5	a79c879d28c5c8a4707d52bbaa57607f	12050	cracked	0:45:41	2006-11-11 09:51:43
md5	a79e1c64d27737e3f959a6a56b41c650		processing3	0:57:18	2006-11-11 09:48:59
md5	2ef5b8b0eee93568a1126bb923664057		processing3	0:57:36	2006-11-11 09:48:41
md5	e53cc072934b25e45dc273c6c342556d		processing3	0:58:7	2006-11-11 09:48:10
md5	d38ad0e58c9525343f492161b87400a1	htmldb	cracked	0:58:23	2006-11-11 09:44:01
md5	d926dbaeb7fac97612ec219f7f172610		processing3	1:4:30	2006-11-11 09:41:47
md5	fcf2483ced17683085849877134fd50c		processing3	1:6:32	2006-11-11 09:39:45
md5	377a8f80271a6f920df0e4aa84d1029a	bombi	cracked	0:43:12	2006-11-11 09:38:26
md5	85d95e2ad51bfcd5d6d352486fbe2769	pupsi	cracked	1:8:2	2006-11-11 09:28:25
md5	96bc2c727049b5dce27bd8b9e8b264bf		processing3	1:19:6	2006-11-11 09:27:11
md5	8aa12bbde69504ba86b942726b4d7623		notfound	1:18:15	2006-11-11 09:02:54
md5	5ce1d809749963448767622e0ca8169f	28264451	cracked	0:48:15	2006-11-11 09:02:35



Roteiro de estudos

1. Leitura das seções 11.1, 11.2, 11.3, 11.5 e 11.6. do livro “Criptografia e segurança de redes. Princípios e práticas”. William Stallings;
2. Estudo da vídeo-aula referente ao tópico 11;
3. Referências complementares:
 - ▶ http://www.serafim.eti.br/academia/recursos/Roteiro_08-Funcoes_de_Hash.pdf

