**PLANO DE AULA PRÁTICA**

**TAREFA 3** "Função Hash"

Tópico no Moodle: “Tarefa 3 – Uso de Funções Hash”

Data: 29-08-2014

Tempo de Aula: 1 horas-aula

Grupo: <Aluno\_1> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 <Aluno\_2> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# ASSUNTO Função Hash

O **controle de integridade** consiste em assegurar-se de que a mensagem recebida é a enviada pela outra parte e não foi manipulada.

Para cumprir com este objetivo utilizam-se funções unidirecionais chamadas funções **hash**.

* **Unidirecionalidade:** conhecido um resumo h(M), deve ser computacionalmente impossível encontrar M a partir do resumo.
* **Compressão:**a partir de uma mensagem de qualquer longitude, o resumo h(M) deve ter uma longitude fixa. O normal é que a longitude de h(M) seja menor do que a da mensagem M.
* **Facilidade de cálculo:**deve ser fácil calcular h(M) a partir de uma mensagem M.
* **Difusão:**o resumo h(M) deve ser uma função complexa de todos os bits da mensagem M: se, se modifica um só bit da mensagem M, o hash h(M) deveria mudar a metade dos seus bits aproximadamente.
* **Colisão simples:**será computacionalmente impossível, conhecido M, encontrar outro M’ tal que h(M) = h(M’). Isto se conhece como resistência débil às colisões.
* **Colisão forte:** será computacionalmente difícil encontrar um par (M, M’) de forma que h(M) = h(M’). Isto se conhece como resistência forte às colisões.

Em um primeiro momento é gerado um resumo criptográfico da mensagem através de algoritmos complexos que reduzem qualquer mensagem sempre a um resumo de mesmo tamanho. A este resumo criptográfico se dá o nome de **hash**.

Uma vez que o hash tem tamanho fixo, deduz-se que o número de funções hash possíveis que podem ser geradas é finito, e em contra partida, o número de mensagens que podem ser geradas e enviadas ser infinitas. De fato, é impossível impedir que mensagens diferentes levem a um mesmo hash. Quando se encontram mensagens diferentes com hashes iguais, é dito que foi encontrada uma colisão de hashes. Um algoritmo onde isso foi obtido deve ser abandonado.



Na transmissão do resumo (hash) pode usar-se também qualquer uma das criptografias já vistas.

1. Uma função que aceita uma mensagem M, de comprimento variável como entrada, e produz **uma saída de comprimento fixo**. Esta é uma função apenas da mensagem M de entrada. O resultado de uma função hash é também conhecido como **resumo de mensagem** ou **valor hash**.

O código hash é uma função de todos os bits da mensagem e oferece a capacidade da detecção de erros: uma mudança em qualquer bit na mensagem, resulta em uma mudança no resultado da função hash. Ver figura 11.5 do livro de Stallings ( pag. 234 e 235).

1. A finalidade de uma função *hash* é produzir uma “impressão digital” (*fingerprint*) de um arquivo, mensagem ou blocos de dados. Funções hash são úteis para autenticação de mensagens e verificação da integridade de mensagens. Com essas funções, pode-se usar o método clássico para se produzir assinaturas digitais. Ver o caso (e) em 11.5.
2. Para ser útil para autenticação de mensagens, uma função hash precisa ter as seguintes propriedades:
3. H pode ser aplicada a um bloco de qualquer tamanho.
4. H produz uma saída de comprimento fixo.
5. H(x) é relativamente fácil de calcular x, tornando as implementações de HW e SW práticas.
6. Para qualquer valor h dado, é computacionalmente inviável encontrar x tal que H(x) = h. Ou seja, H é uma função que não tem a sua função inversa H-1. Isso é conhecido como a **propriedade unidirecional** da função hash.
7. Para qualquer bloco de dado x, é computacionalmente inviável encontrar outro bloco de dado y diferente de x, tal que H(y) = H(x). Isso é conhecido como **resistência fraca a colisões**.
8. É computacionalmente inviável encontrar qualquer par (x,y) tal que H(x) = H(y). Isso é conhecido como **resistência forte a colisões**. Ver o exemplo do ataque do aniversário, entregue em sala de aula.

**BIBLIOGRAFIA BÁSICA**

<http://www.gta.ufrj.br/grad/09_1/versao-final/assinatura/hash.htm>

Livro texto: Criptografia e Segurança de Redes, Stallings.

 **OBJETIVOS DA AULA**

Conhecer um algoritmos de cálculo de funções Hash.

**SELEÇÃO DE CONTEÚDO – A PRÁTICA**

**Solução Java 1:**

Uma maneira de se gerar um código hash é usar Java é utilizando a classe **java.security.MessageDigest**. O método a seguir gera o hash MD5 de uma string:

MessageDigest md = MessageDigest.getInstance("MD5");

md.update(frase.getBytes());

byte[] hashMd5 = md.digest();

O hash é retornado como um array de bytes, mostrado no método

private static String stringHexa(byte[] bytes) {

 StringBuilder s = new StringBuilder();

 for (int i = 0; i < bytes.length; i++) {

 int parteAlta = ((bytes[i] >> 4) & 0xf) << 4;

 int parteBaixa = bytes[i] & 0xf;

 if (parteAlta == 0) s.append('0');

 s.append(Integer.toHexString(parteAlta | parteBaixa));

 }

 return s.toString();

}

Além do MD5, podemos também usar outros algoritmos de hash normalmente presentes nas distribuições da JVM. O exemplo a seguir gera os códigos MD5, SHA-1 e SHA-256 da mesma mensagem:

public static byte[] gerarHash(String frase, String algoritmo) {

 try {

 MessageDigest md = MessageDigest.getInstance(algoritmo);

 md.update(frase.getBytes());

 return md.digest();

 } catch (NoSuchAlgorithmException e) {

 return null;

 }

}

public static void main(String[] args) {

 String frase = "Quero gerar códigos hash desta mensagem.";

 System.out.println(stringHexa(gerarHash(frase, "MD5")));

 System.out.println(stringHexa(gerarHash(frase, "SHA-1")));

 System.out.println(stringHexa(gerarHash(frase, "SHA-256")));

}

A saída deste programa será:

51408e7592917e916b55f8021a0f7ad3 (MD5 128 bits)

5907736c4de6fdd70fcdc7517c3aebd257a1addc (SHA-1,128 bits)

ed40c896d779f946956483d45e9efe08aac6373935d354c2be2136bf05bca42c (SHA-2, 256 bits)

---------------------------------------------------------------------------------------------------

**Solução Java 2:**

Java possui o pacote java.security.\*. E assim, podemos obter uma impressão digital a partir de uma mensagem. A classe **MessageDigest** é, de fato, uma classe abstrata, mas podemos continuar obtendo uma instância dela, usando o método **getInstance()**. Neste método, incluímos uma *string*, indicando o algoritmo que gostaríamos de usar. Quando o nosso programa já tiver uma instância de **MessageDigest**, ele poderá começar a leitura de nossa mensagem, um byte por vez, ou como um *array* de bytes. Quando todos os bytes da mensagem tiverem sido lidos , o método digest() poderá ser chamado para invocar o algoritmo de *hash* e retornar uma impressão digital (resumo de mensagem). Veja o programa Java:

import Java.security.\*;
public class Fingerprint {
 public static void main(String [] args){
 MessageDigest md = null;
 String message = “”;
 for (int i=0; i<args.length;i++)
 message = message + “ “ + args[i];
 try {
 md = MessageDigest.getInstance(“SHA-1”);

 } catch (NoSuchAlgoritmException ae) {}

 md.update( message.getBytes());
 byte [] fingerprint = md.digest();

 System.out.print(“Fingerprint: “);
 for(int j=0; j<fingerprint.length; j++)
 System.out.print( (fingerprint[j] + 128) + “ “);
 }
}

Execute este programa, entrando com uma mensagem M, usando argumentos de linha de comando. Anote seu *hash*. Entre com a sua mensagem M’, ligeiramente alterada em M. O que você observa, comparando os valores *hash* calculados ?

-------------------------------------------------------------

Ou use a ferramentas que fornecem cálculos de Funções Hash:

* HashCalc
http://www.slavasoft.com/hashcalc/index.htm
* ADLER 32 HASH CALCULATOR

http://www.md5calc.com/adler32

**QUESTÕES**

1. Para que serve, em segurança, o uso das funções *hash* ?
2. Vamos supor que uma mensagem M chegue a você e que um *hash* da mensagem H(M) (resumo da mensagem ou impressão digital) chegue **separadamente**. Pode-se afirmar que a verificação de integridade da mensagem é segura ?
3. Qual é a deficiência para este método de verificação de integridade ?

Uma deficiência para este método de verificação está relacionada à impressão digital H(M), que foi enviada **separadamente** da mensagem M.

1. Qual a causa do problema de segurança existente ?

 Os algoritmos SHA-1, MD 5 ou outros (SHA-256, SHA-384, SHA-512)
 estão disponíveis publicamente e, portanto, se alguém interceptar
 a mensagem M e o H(M), poderá modificar a mensagem M para M’,
 gerar um nova impressão digital H(M’) e enviar ambas para você.
 Você não poderá supor que a mensagem M tenha sido alterada.

1. Existem alguns algoritmos disponíveis, mais usados, para hash de mensagens: MD5 (128 bits, ainda considerado útil) e o SHA-1(160 bits, considerado excelente). Normalmente, os sites que fornecem arquivos longos para serem transferidos, mencionam estes algoritmos e fornecem os cálculos dos *hash* para esses arquivos. Existe alguma vantagem nesta forma de fornecimento dos cálculos de *hash* ?
2. O que pode ser feito para amenizar o problema de envio separado de hashes e mensagens ?

**Código de Autenticação de Mensagens (em inglês, MAC)** - Veja slides na página da disciplina.

**MOSTRE SEU EXPERIMENTO**

Ao final da aula, deposite sua tarefa no Moodle, no tópico Tarefa 3.

O Moodle será fechado a partir de 22:00 na data desta aula.
É a comprovação de sua participação na Tarefa 3 da disciplina.