Insper

Camada Física da Computação

Aula 26 – CRC e funções de hash

2016 – Engenharia

Fábio Ayres <fabioja@insper.edu.br>

Objetivos

- Calcular o CRC de uma string binária
- Conhecer algumas das funções de hash criptográficas para detecção de erros

Cyclic redundancy check (CRC)

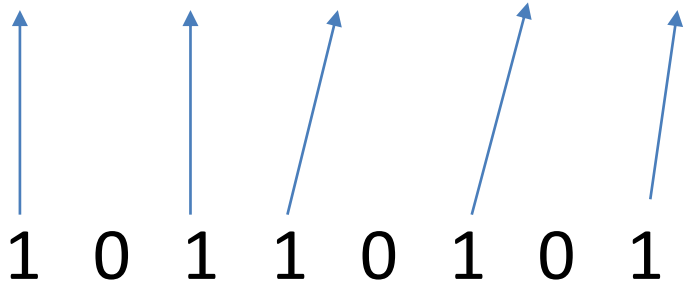
Este código se baseia em divisão de polinômios:

- Interpretar os dados como sendo coeficientes de um polinômio
- Cada código CRC tem um polinômio especial a ser usado como divisor
- O resto da divisão forma os bits de redundância do código

Funcionamento do CRC

- Dados como polinômios: se os bits que queremos mandar forem 10110101, o polinômio correspondente é

$$x^7 + x^5 + x^4 + x^2 + x^0$$



Insper

Funcionamento do CRC

- Adotar um polinômio de CRC (existem vários)
- Exemplo: $\mathrm{CRC}{-}4 = x^4 + x + 1$ (10011)

Funcionamento do CRC

- Para um polinômio de CRC de grau r, adicionar r zeros ao final dos dados

- Equivalente a multiplicar o polinômio dos dados por x^r

$$\begin{cases} \text{dados: } 10110101 \\ \text{polinômio CRC: } 10011 \end{cases} \rightarrow 10110101\mathbf{0000}$$

Funcionamento do CRC

- Efetuar a divisão polinomial, com uma mudança: a aritmética da adição e subtração é substituída pelo operador XOR

```
  1011
- 1100
  ----
  0111
```

Exemplo

```
101101010000 |10011
```

Exemplo

```
  1011010100000 |10011
- 10011          1
  -----
  001011010000
```

Subtração XOR

Se o bit mais alto for 1, fazer o XOR dos dados com o polinômio CRC

Exemplo

```
  1011010010000 |10011
- 10011          10
  -----
  0010110010000
-  00000
   -----
  001011010000
```

Se o bit mais alto for 1, não precisa fazer nada

Exemplo

```
  101101010000 |10011
- 10011         10101010
  -----
  001011010000
 - 00000
  -----
  001011010000
  - 10011
  -----
  000010110000
  - 00000
  -----
  000010110000
  - 10011
  -----
  000000101000
  - 00000
  -----
  000000101000
  - 10011
  -----
  000000001110
  - 00000
  -----
  000000001110
```

Ignorar o quociente
(não precisa nem guardar)

Este é o CRC

Detecção de erro com CRC

- Enviar os dados junto com o CRC

101101011110

- Na recepção, recalcular a divisão polinomial dos dados recebidos, incluindo os bits de CRC

- Se o resultado final for zero, não houve erro na transmissão

Vantagens do CRC

- CRC-16 e CRC-CITT
  - 100% das falhas em sequências de 16 ou menos bits
  - 99.997% das falhas em sequências de 17 bits
  - 99.998% em sequências de 18 bits ou mais
- CRC-32
  - Chance de receber dados ruins é de 1 em 4.3 bilhões

Fonte: http://www.inf.ufrgs.br/~asc/redes/pdf/aula06.pdf

Exercício

- Queremos transmitir a sequência 110101000 usando o polinômio CRC 1001. Qual a sequência binária a ser enviada?
- Suponha que a sequência recebida no exemplo anterior não contém erros. Aplique o procedimento de cálculo do CRC na sequência recebida e verifique que o resultado é zero.
- Suponha que o quarto bit mais significativo da sequência sofreu um erro. Repita o procedimento de CRC para ver que haverá detecção de erro.
- (Mega desafio) Implemente uma função que calcule o CRC de uma string qualquer.
- (Ultra mega desafio) Implemente uma função eficiente que calcule o CRC de uma string qualquer! (Use look-up tables)

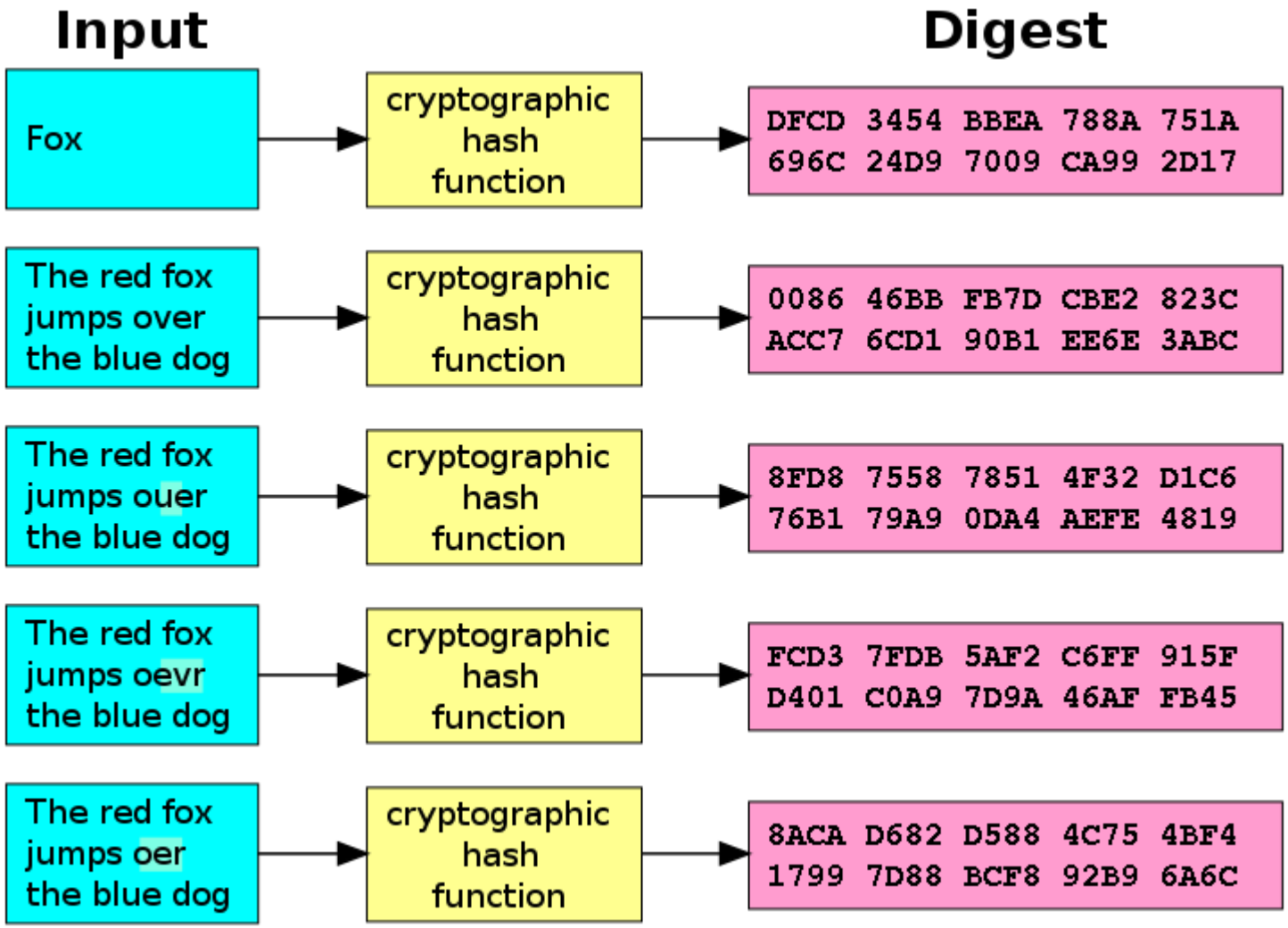
Funções de hash

São funções que...

- ...transformam uma mensagem (uma string binária de comprimento arbitrário)...
- ...em uma string binária de tamanho fixo (chamada de *hash* ou *message digest*)...
- ...com algumas propriedades (continua no próximo slide...)

Propriedades de funções de hash

- Calcular o hash de uma mensagem é uma operação rápida
- É inviável computar a mensagem original a partir de seu hash, a não ser tentando todas as possibilidades
- Uma pequena mudança na mensagem deve resultar em um novo valor de hash que é completamente decorrelacionado do valor de hash antigo
- É inviável encontrar duas mensagens diferentes com o mesmo valor de hash
  - Esse fenômeno chama-se *colisão*
  - Não significa que não exista colisão, apenas que é difícil encontrá-la.



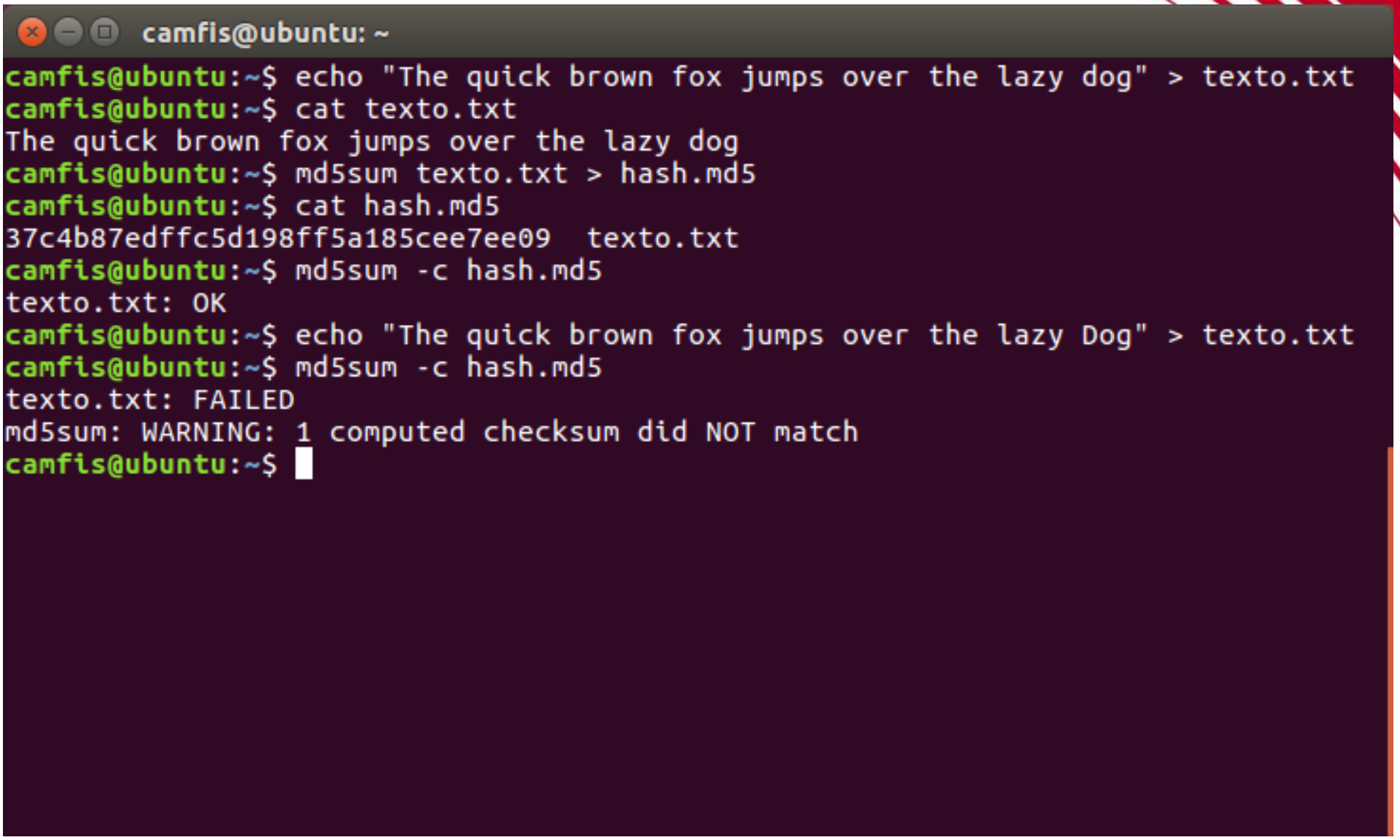
Fonte: https://en.wikipedia.org/wiki/Cryptographic_hash_function

Algumas funções famosas

- MD5
- SHA-1
- SHA-3

MD5

- Desenvolvido por Ronald Rivest (MIT) em 1992
  - Inventor do RSA, fundador da Verisign, co-autor de um dos livros mais famosos de algoritmos, entre outras coisas.
- Um dos métodos mais populares de geração de *message digests*.
- Transforma mensagens de comprimento qualquer em um hash de 128 bits



Nota: hash diferente da wikipedia por causa do caractere de fim de arquivo

Vulnerabilidades

- O MD5 não é mais considerado criptograficamente seguro!
  - Não deve mais ser usado para assinaturas digitais, somente para verificação de erros em situações sem problemas de segurança

- Para casa: procure sobre o vírus *Flame*, que explorou colisões do MD5 para falsificar a assinatura de certificação do Windows.

SHA-1

- Secure Hash Algorithm – 1
- Projetado pela NSA, divulgado em 1995
- Produz valor de hash de 160 bits (20 bytes)
- Você já usa SHA-1 o tempo todo: usado para assinar commits do Git!

Vulnerabilidades do SHA-1

- “SHA-1 is no longer considered secure against well-funded opponents. In 2005, cryptanalysts found attacks on SHA-1 suggesting that the algorithm might not be secure enough for ongoing use, and since 2010 many organizations have recommended its replacement by SHA-2 or SHA-3. Microsoft, Google and Mozilla have all announced that their respective browsers will stop accepting SHA-1 SSL certificates by 2017.”
- https://en.wikipedia.org/wiki/SHA-1


camfis@ubuntu: ~
camfis@ubuntu:~$ echo "The quick brown fox jumps over the lazy dog" | sha1sum
be417768b5c3c5c1d9bcb2e7c119196dd76b5570 -
camfis@ubuntu:~$ echo "The quick brown fox jumps over the lazy Dog" | sha1sum
f4756c80b8c91564576ae7682ded87fe2b5be926 -
camfis@ubuntu:~$ echo "The quick brown fox jumps over the lazy dog" | sha512sum
a12ac6bdd854ac30c5cc5b576e1ee2c060c0d8c2bec8797423d7119aa2b962f7f30ce2e39879cbff
0109c8f0a3fd9389a369daae45df7d7b286d7d98272dc5b1 -
camfis@ubuntu:~$

Exercício

- Crie um arquivo de texto e calcule seu hash usando md5sum, sha1sum e sha512sum

Outras aplicações de funções de hash

- Existem muitas outras funções de hash, e nem todas as aplicações requerem robustez criptográfica

- Funções de hash tem MUITA aplicação em algoritmos

Outras aplicações de funções de hash

- *hashmaps* (ou *hashtables*): estruturas do tipo chave-valor (como dicionários do Python) que permitem busca de chaves em tempo médio constante, ou seja O(1)
  - "Hashtables: Arguably the single most important data structure known to mankind. You absolutely should know how they work. Be able to implement one using only arrays in your favorite language, in about the space of one interview." – email pré-entrevista do Google.

Outras aplicações de funções de hash

- Bloom filters: permitem determinar se um item está em um conjunto (com certa probabilidade) ou não (com certeza)
  - Muito usado em bancos de dados e máquinas de busca
- Sharding
  - Balanceamento de carga de dados ou de processamento
  - Divisão de dados baseada no hash de cada chave

Insper

www.insper.edu.br