Autenticação de mensagens

No universo digital uma necessidade premente é autenticar mensagens, onde mensagem aqui pode significar qualquer objeto digital (e binário): imagens, arquivos, mensagens trocadas, objetos que sofreram upload ou dowmload, etc etc.

As funções que fazem esta autenticação recebem o nome de *funções hash* e *hash* em inglês significa picar, cortar miúdo, triturar. Uma função hash é aplicada sobre uma mensagem (não há chaves envolvidas) e e ela gera uma assinatura do arquivo. Esquematicamente:$A = hash(M)$ As características da função *hash* são:

- M pode ter qualquer tamanho ($0 \leq len(M) < \infty$)
- A função de hash é irreversível. Isto é, dado M é fácil calcular A, mas dado A é computacionalmente inviável achar M.
- A sempre tem tamanho fixo, usualmente medido em bits.
- hash deve ter efeito cascata: Mudando 1 bit em M, espera-se que mude a metade dos bits em A e de maneira imprevisível.
- A função $hash()$ é conhecida, publicada (e testada).
- A busca de A sem conhecer M só pode se dar por força bruta (Aliás, na arquitetura bitcoin, é exatamente isto que caracteriza o $proof-of-work$ lá).
- Nada impede de implementar o $hash()$ em hardware especializado (de novo, na arquitetura bitcoin, é exatamente isto que se faz nas estações especializadas de mineração).

A é conhecido como sumário de M, pois não deixa de ser um resumo de M.

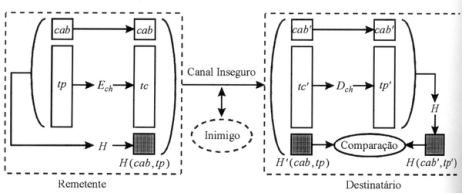
Uma boa função de hash, garante que o "inimigo" não conseguirá:

i obter informações de M a partir de A.
ii Criar um substituto de M a partir do conhecimento de A
iii Achar dois objetos M_1 e M_2 distintos que gerem ambos a mesma A

Ou seja, em termos abstratos, espera-se que uma função H_1 que gera uma assinatura A_1 de comprimento n_1 bits tenha uma dificuldade computacional para ser quebrada maior do que 2^{n_1} Na busca de i e ii acima.

Já para o critério iii a complexidade esperada é apenas $2^{n_1/2}$ já que a busca de qualquer par M_1 e M_2 é muito mais simples do que a busca de um par específico.

Em termos conceituais e especificamente no contexto de criptografia as funções hash podem ser estudadas na figura abaixo:



O remetente quer criptografar e enviar a mensagem t_p para o destinatário. Ele cria o cabeçalho cab e a mensagem t_p calculando o hash sobre o bloco completo. Depois ele criptografa t_p usando a chave ch gerando t_c e monta a mensagem a transmitir: cab, t_c e $hash$ enviando-a por um canal inseguro. [1] O destinatário ao receber a mensagem decriptografa t'_c obtendo t'_p. além de cab' e $hash'$. O que ele precisa fazer agora é calcular o hash de $cab' + tp'$. Comparando este hash com o $hash'$ recebido pode certificar que a mensagem não foi alterada pelo possível inimigo.

Em um contexto mais simples, pode-se usar a função hash simplesmente para garantir que um texto plano qualquer não foi alterado na sua manipulação. O esquema bitcoin usa ambos os contextos acima (criptografia ou simples transmissão).

[1] t_p significa texto plano e t_c texto criptografado

Famílias de funções *hash* Existem inúmeras funções de hash. Você pode inclusive criar a sua função, embora isto não é recomendado por mais capaz que você seja. Isso, porque as funções consagradas já foram exaustivamente testadas sendo provadas como fortes contra ataques de criptoanálise. O uso da estratégia *olhe mamãe, agora sem as mãos* pode acabar com muitos dentes quebrados...

Historicamente uma das primeiras a ser usada em larga escala foi a família MD que começou com o MD_2 que foi substituído pelo MD_4 em 1990 e depois pelo MD_5 em 1992. Todos da autoria de R. Rivest do MIT e da RSA. Todos geram assinaturas de 128 bits. O MD_4 foi desenvolvido para ser mais rápido que o MD_2 e o MD_5 veio resolver algumas deficiências do 4 (Por exemplo, foram encontradas colisões com apenas 2^{20} operações). Apenas para conhecimento eis o algoritmo do MD_5

```
Completa-se a entrada com zeros em múltiplo de 64 bytes
A ← 6745.2301
B ← EFCD.AB89
C ← 98BA.DCFE
D ← 1023.5476
para para L de 1 até M
   (A', B', C', D') ← (A,B,C,D)
   o bloco de 512 vira 16 palavras de 32 bits (x_k)
   j ← 1
   para i de 1 a 16
      T ← A+F(B,C,D)+x_ki+Y_j
      (A,D,C,B) ← (D,C,B,B+(T≤ 2s_i))
      j ← j+1
   fim{para}
   para i de 1 a 16
      T ← A+G(B,C,D)+x_ki+Y_j
      (A,D,C,B) ← (D,C,B,B+(T≤ 2s_i))
      j ← j+1
   fim{para}
   para i de 1 a 16
      T ← A+H(B,C,D)+x_ki+Y_j
      (A,D,C,B) ← (D,C,B,B+(T≤ 2s_i))
      j ← j+1
   fim{para}
   para i de 1 a 16
      T ← A+I(B,C,D)+x_ki+Y_j
      (A,D,C,B) ← (D,C,B,B+(T≤ 2s_i))
      j ← j+1
   fim{para}
   (A,B,C,D) ← (A+A', B+B', C+C', D+D')
fim{para}
sumário ← (A,B,C,D)
```

Outra família é a SHA desenvolvida pelo governo norte americano em 1994. O SHA_1 gera um sumário de 160 bits e seus sucessores foram aumentando este valor. No Bitcoin, usa-se a função $SHA256$ que gera um sumário de 256 bits. Em python3 esta função (SHA_1) é obtida fazendo-se

```
>>> import hashlib
>>> hashlib.sha1(b'bloco...').hexdigest()
>>> hashlib.sha1(b'').hexdigest()
'da39a3ee5e6b4b0d3255bfef95601890afd80709'
```

Como citado as duas implementações do SHA que são usadas na arquitetura Bitcoin são $SHA256$ e $RIPEMD160$ que podem ser assim chamadas:

```
>>> import hashlib
>>> hashlib.sha256(b'conteudo...').hexdigest()
>>> h = hashlib.new('ripemd160')
>>> h.update(b'conteudo...')
>>> h.hexdigest()
>>> sha_256('...') # programada p/ usuário...
```

☞ Para você fazer

A seguir uma implementação *toy domain* para você vivenciar a implementação real. Outra razão de ser simplificada é para você não ter desejos de usá-la em processamento real: NÃO É UMA BOA IDÉIA.

1. Considere um alfabeto composto pelos seguintes caracteres:

```
(/): 0   F : 6   L :12   R :18   X :24
 A : 1   G : 7   M :13   S :19   Y :25
 B : 2   H : 8   N :14   T :20   Z :26
 C : 3   I : 9   O :15   U :21   ! :27
 D : 4   J :10   P :16   V :22   ? :28
 E : 5   K :11   Q :17   W :23   * :29
```

2. Note que o alfabeto acima tem base=30
   - Para converter conjunto de 4 caracteres (L4L3L2L1) em número faça:
     NUMERO $= L_4 \times 30^3 + L_3 \times 30^2 + L_2 \times 30 + L_1$, ou
     NUMERO $= (L_4 \times 27000) + (L_3 \times 900) + (L_2 \times 30) + L_1$
     Exemplo: ABCD $= 1 \times 27000 + 2 \times 900 + 3 \times 30 + 4 = 28894$
   - Para converter um número N ($N \leq 810000$) em L4L3L2L1, faça:
     $L_4 =$ ALFABETO[$A_1 \leftarrow$ N div 27000]
     $L_3 =$ ALFABETO[$A_2 \leftarrow$ (N - (27000 × A1)) div 900]
     $L_2 =$ ALFABETO[$A_3 \leftarrow$ (N - ((27000 × A1)+(900xA2)) div 30]
     $L_1 =$ ALFABETO[N-((27000× A1)+(900 × A2)+ (30 × A3))]
3. Obtenha o arquivo para o qual se quer obter a assinatura
4. Aplique o algoritmo abaixo ao arquivo

Este algoritmo recebe um arquivo qualquer que só contém os caracteres acima, calcula e devolve a assinatura (16 caracteres).

1. Divida o arquivo original em blocos de 12 caracteres. Preencha o ultimo com espaços (/) se necessário. Calcule m ← número de blocos
2. Defina as seguintes variáveis: A ← 499901 (corresponde a `ROMK`)
   B ← 377007 (corresponde a `M?Z!`)
   C ← 757331 (corresponde a `?ANK`)
   D ← 111931 (corresponde a `DDKA`)
3. Defina as funções $F(x, y, z)$ e $G(x, y, z)$ como sendo $F(x, y, z) = (x + y)\ mod\ z$
   $G(x, y, z) = (2 \times valor\ absoluto(x - y))\ mod\ z$
4. Aplique:

```
1: para j de 1 até m
2:    para k de 1 até 3
3:       X_k ← número correspondente a k-ésima parte do bloco j
4:       T ← (A + F(B,C,D) + Xk + tamoriginal(arquivo)) mod 809999
5:       (A,D,C,B) ← (D,C,B,T)
6:    fim{para}
7:    para k de 1 ate 3
8:       X_k ← número correspondente a k-ésima parte do bloco j
9:       T ← (A + G(B,C,D) + Xk + tamoriginal(arquivo)) mod 809999
10:      (A,D,C,B) ← (D,C,B,T)
11:   fim{para}
12: fim{para}
```

5. O tamanho do objeto NÃO considera os eventuais espaços incluídos ao final.
6. A assinatura do arquivo é A,B,C,D

Veja este código em `f:/p/n/n05/fun.py`. Para testar o funcionamento deste algoritmo, tem-se

```
Assinat('CURITIBA/PARANA') = MZQ?OIG!NRNSIYF!
Assinat('CORITIBA/PARANA') = QDULPO?OLN**PZ/Y
Assinat('A')               = /IV!JHLZ?QBSDQGA
Assinat('AA')              = /KWAJLUS?LBGDPF?
Assinat('AAA')             = /KYEJOUE?KZYDPEY
```

Calcule a assinatura da frase

```
D O S / V A R I O
```