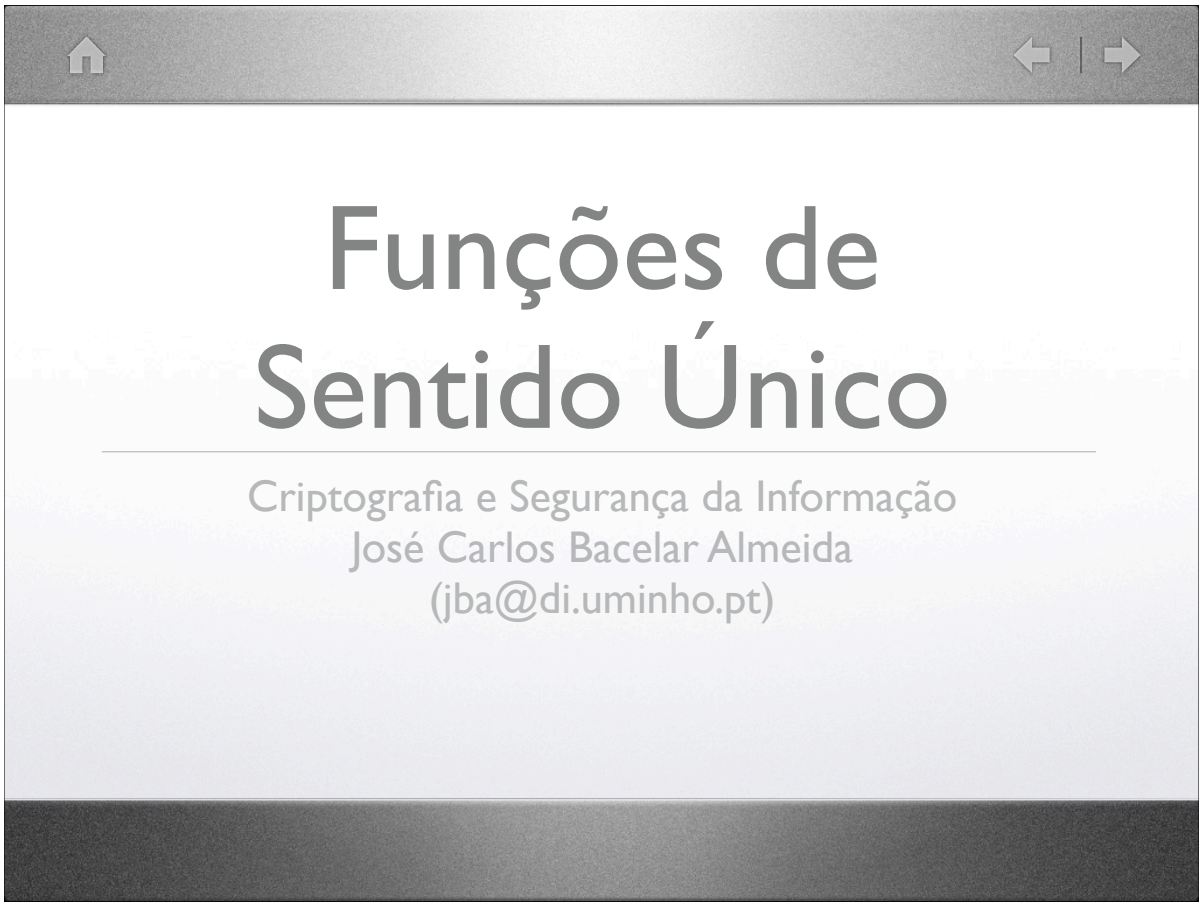
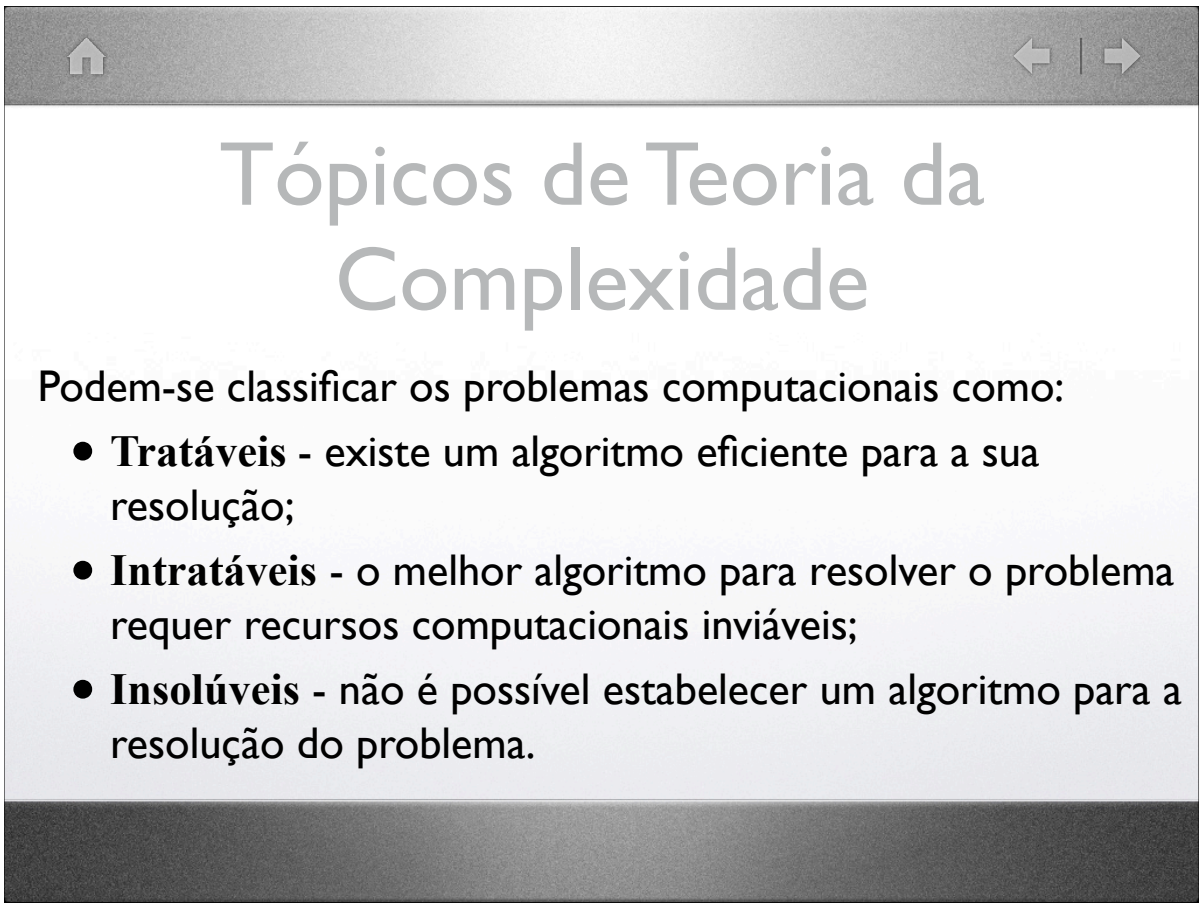


Funções de Sentido Único

Criptografia e Segurança da Informação
José Carlos Bacelar Almeida
(jba@di.uminho.pt)



Tópicos de Teoria da Complexidade

Podem-se classificar os problemas computacionais como:

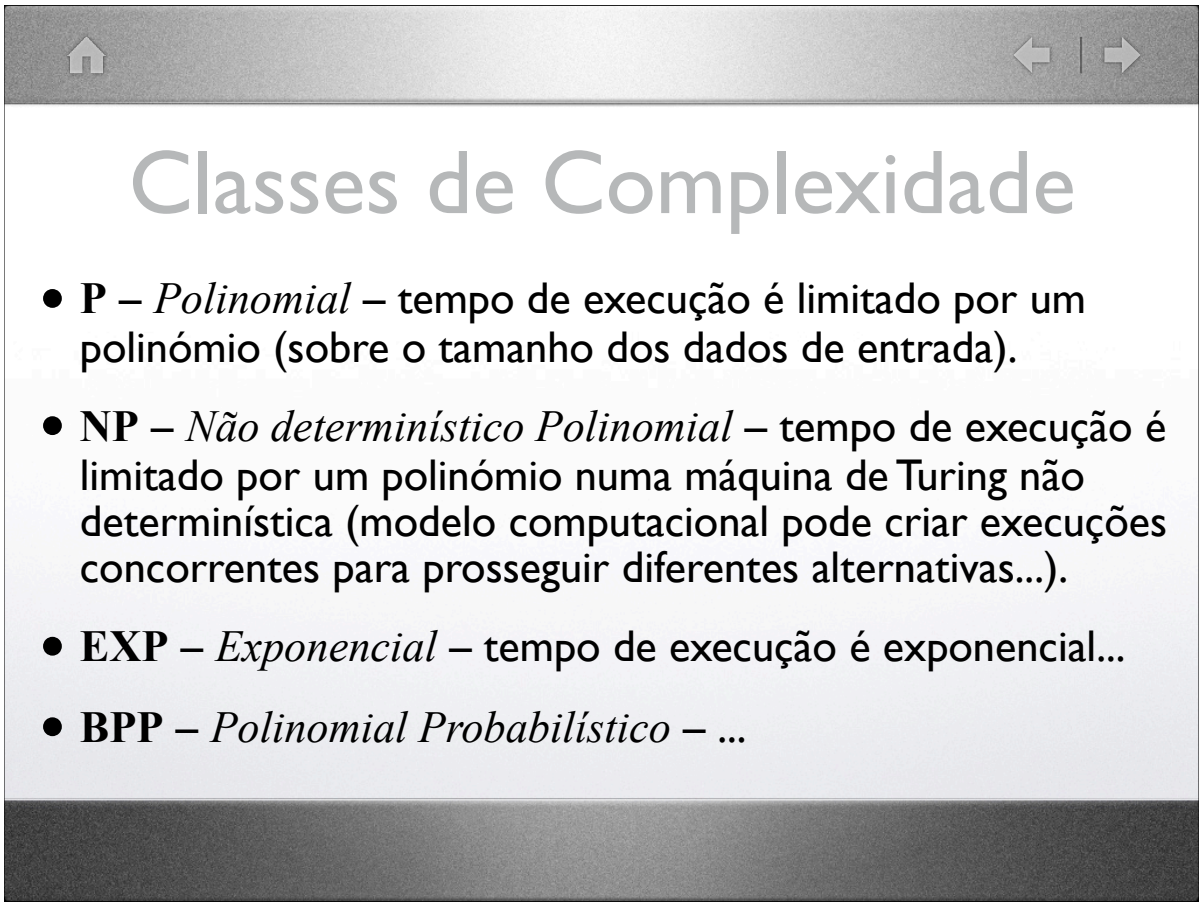
- **Tratáveis** - existe um algoritmo eficiente para a sua resolução;
- **Intratáveis** - o melhor algoritmo para resolver o problema requer recursos computacionais inviáveis;
- **Insolúveis** - não é possível estabelecer um algoritmo para a resolução do problema.

Computabilidade

- Início do século XX...
- Formaliza a noção de algoritmo.
- Diferentes modelos computacionais determinam a mesma noção de problema solúvel (e.g. λ-calculus; máquinas de Turing; máquinas de registos; linguagem *While*).
- Exemplos de problema insolúveis (indecidíveis): *halting problem*; reconhecimento de linguagens tipo 0; problema de correspondência de Post.

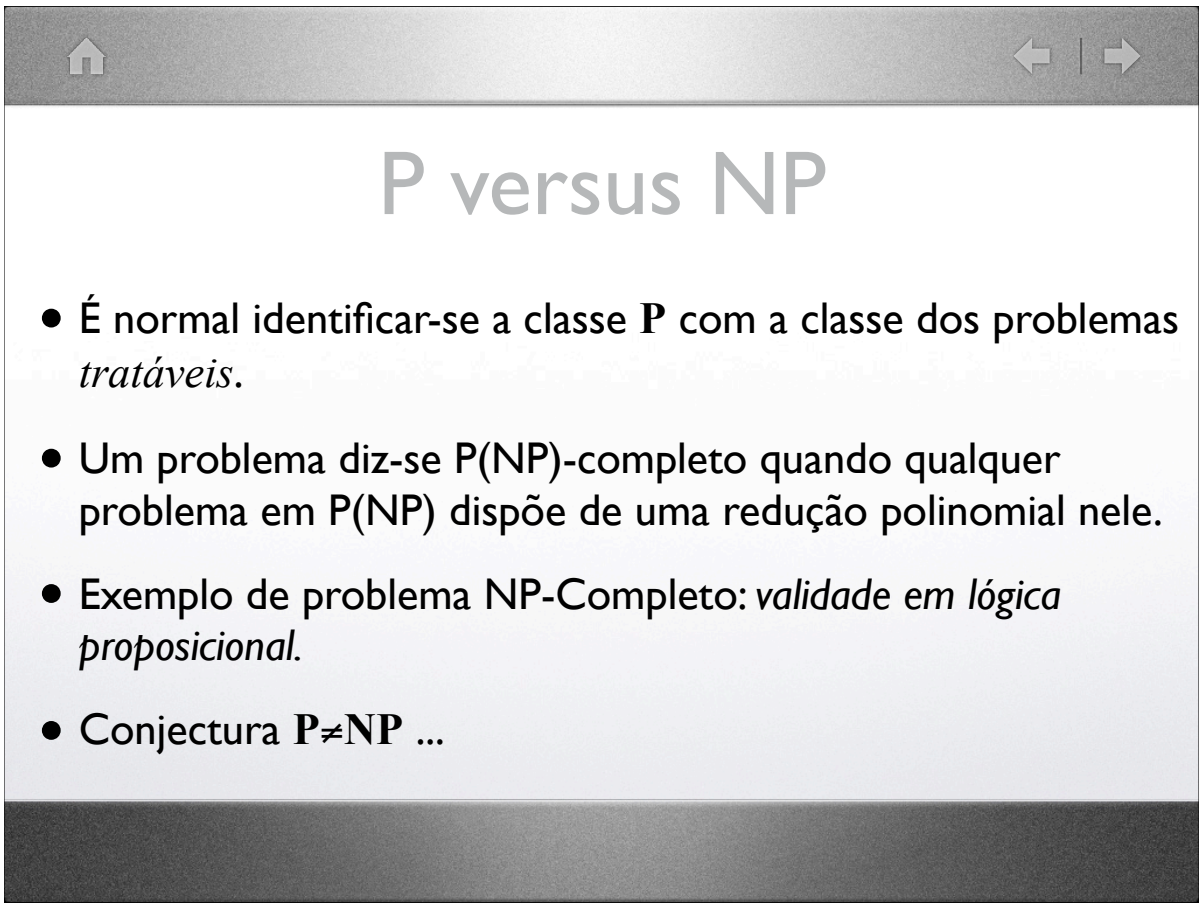
Complexidade

- Procura avaliar a eficiência (temporal e/ou espacial) dos algoritmos em função do "tamanho" dos dados de entrada.
- Modelo computacional adoptado é normalmente a máquina de Turing (mas...)
- É normal conduzirmos a análise para o *pior caso*, mas também é possível considerar o *caso médio* (análise estatística).
- Análises assimptótica (e.g. $O(2^n)$; $O(n^5)$)



Classes de Complexidade

- **P** – *Polinomial* – tempo de execução é limitado por um polinómio (sobre o tamanho dos dados de entrada).
- **NP** – *Não determinístico Polinomial* – tempo de execução é limitado por um polinómio numa máquina de Turing não determinística (modelo computacional pode criar execuções concorrentes para prosseguir diferentes alternativas...).
- **EXP** – *Exponencial* – tempo de execução é exponencial...
- **BPP** – *Polinomial Probabilístico* – ...



P versus NP

- É normal identificar-se a classe **P** com a classe dos problemas *tratáveis*.
- Um problema diz-se P(NP)-completo quando qualquer problema em P(NP) dispõe de uma redução polinomial nele.
- Exemplo de problema NP-Completo: *validade em lógica proposicional.*
- Conjectura **P≠NP** ...

Funções de sentido único

- Em criptografia gostaríamos de dispor de funções que:
  - possuam um algoritmo eficiente para o seu cálculo,
  - não disponham de um algoritmo eficiente que calcule uma sua (pseudo-)inversa.

essas funções são designadas por

Funções de Sentido Único

FSU (cont.)

- É óbvio que funções de sentido único com domínio finito pertencem à classe NP.
- Mas:
  - **P≠NP** não implica a existência de funções de sentido único.
  - Complexidade do "pior caso" não é adequada para garantir segurança.
  - Mas *acredita-se* que certas funções cumprem os requisitos (a ***teoria dos números computacional*** tem-se revelado a principal fonte para esses problemas).

Funções de Hash criptográficas

- Um exemplo de aplicação de funções de sentido único são a *funções de hash criptográficas*.
- A sua segurança baseia-se, portanto, em argumentos de natureza de *complexidade computacional*.
- O objectivo é que mensagens de comprimento arbitrário sejam mapeadas num contra-domínio de tamanho fixo.
- ... mas devem ser "de sentido único" no sentido em que não deve ser possível inverter essa função.
- Exemplos: MD5, SHA-1, RIPEMD-160

Propriedades

Os requisitos das funções de hash são normalmente expressos pelas propriedades:

- ***(First) pre-image resistant***: dado um valor de hash *h*, deverá ser inviável conseguir obter uma mensagem *m* tal que *hash(m)=h*.
- ***Second pre-image resistant***: dada uma mensagem m_1, deverá ser inviável obter uma mensagem m_2 distinta de m_1 tal que $hash(m_2)=hash(m_1)$.
- ***Collision resistant***: não é viável encontrar mensagens distintas m_1 e m_2 tais que $hash(m_1)=hash(m_2)$.

A *resistência a colisões* é a propriedade que se deseja nas funções de hash criptográficas. No entanto, em certas aplicações é suficiente uma das propriedades mais fracas.

Birthday attack

- Um resultado famoso da teoria das probabilidades indica-nos que necessitamos de um contra-domínio de "tamanho razoável" para se conseguir resistência a colisões.

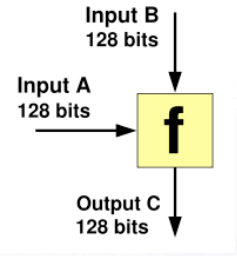
  Quantas pessoas se tem (em média) que perguntar a idade numa festa de anos para encontrar duas com o mesmo dia de aniversário?

  ...testando cerca de $sqrt(N)$ valores aleatórios do domínio dispõe-se de probabilidade superior a ½ de encontrar uma colisão!
- Valores típicos para contra-domínios de funções de Hash criptográficas: 128..512 bit.
- ...assim, um ataque por força bruta para encontrar colisões deve testar entre 2^{64} e 2^{256} mensagens.

Aplicações das funções de hash criptográficas

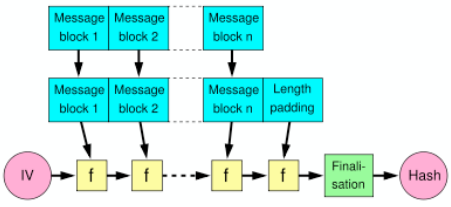
- Armazenamento de *passwords*.
- *Commitment schemes* (provas de "posse" de informação).
- Amplificação de *Entropia* (e.g. Password-based Key Derivation Functions)
- como componentes de outras técnicas:
  - MACs
  - Geradores de sequências aleatórias seguras (PRNG)
  - Cifras
  - ...

Desenho de Funções de Hash Criptográficas



- Um ingrediente fundamental no desenho de funções de hash criptográficas são as **funções de compressão**:
  - Estas são funções de sentido único nos seguintes sentidos:
    - conhecendo ambas as entradas, é fácil calcular a saida;
    - conhecendo a saída, é difícil calcular qualquer uma das entradas;
    - conhecendo a saída e uma das entradas, é difícil calcular a outra.
  - Devem também ser *resistentes a colisões*.
- Podem ser construídas a partir de cifras por blocos... (cumprem metade dos requisitos directamente - existem construções standard que permitem obter funções de compressão a partir dessas cifras)

- A generalidade das funções de hash baseia-se na construção de *Merkle-Damgård*:



- A função de compressão é responsável por fazer evoluir o estado interno (do estado anterior e de um bloco da mensagem).
- O IV é normalmente específico do algoritmo (constante).
- Demonstra-se que, se *f* é uma função de compressão livre de colisões, a função de hash resultante também o é.
- É importante o *padding* conter informação relativa ao comprimento da mensagem.

MD5

- Proposto pela *RSA Labs* (Donald Rivest).
- Melhoramento da função MD4.
- Tamanho do contra-domínio: 128 bit.
- Processa a mensagem em blocos de 512 bit.
- Opera por *rounds* (4).
- Nos últimos anos tem surgido avanços importantes na sua cripto-análise. Em particular, já foram encontradas colisões.
- É por isso desaconselhada para aplicações com requisitos de segurança elevada.

MD5 (cont.)

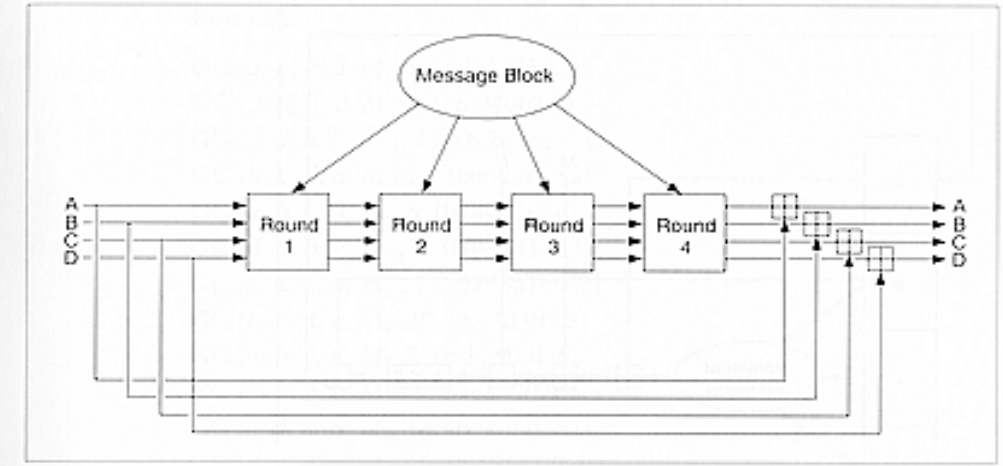
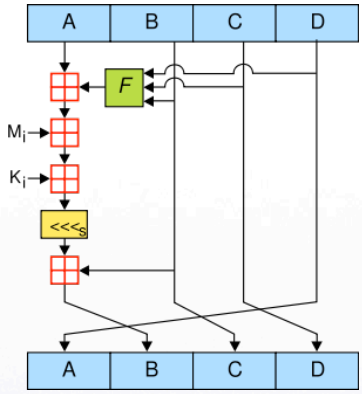


Figure 18.5 MD5 main loop.

- Cada *round* itera a seguinte operação 16 vezes:



  - *A*, *B*, *C* e *D* formam o estado do algoritmo (32 bit cada, inicializadas com valores pré-definidos);
  - *F* é uma função não linear (distinta em cada *round*);
  - M_i é um sub-bloco de 32 bit (M_0-M_{15});
  - *K*i são valores definidos pelo algoritmo (64 valores de 32 bit).

SHA-1

- Função de hash incluída no DSS (Digital Signature Standard)
- Desenvolvida pela NSA (para a NIST) – 1993/1995.
- ...também um melhoramento (correcção) de uma versão anterior (SHA-0).
- Tamanho do contra-domínio: 160 bit.
- Optimizada para arquitecturas *big-endian.*
- Sua cripto-análise também tem sido alvo de avanços recentes significativos.
- Já foram propostas variantes mais seguras (SHA-2: sha-224, sha-256, sha-384, sha-512)

- A,B,C,D,E: 32 bit
- 4 rounds
- 20 operações/round

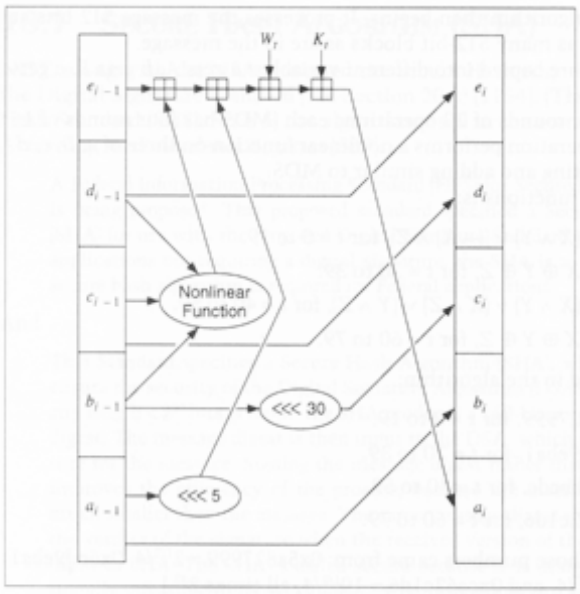
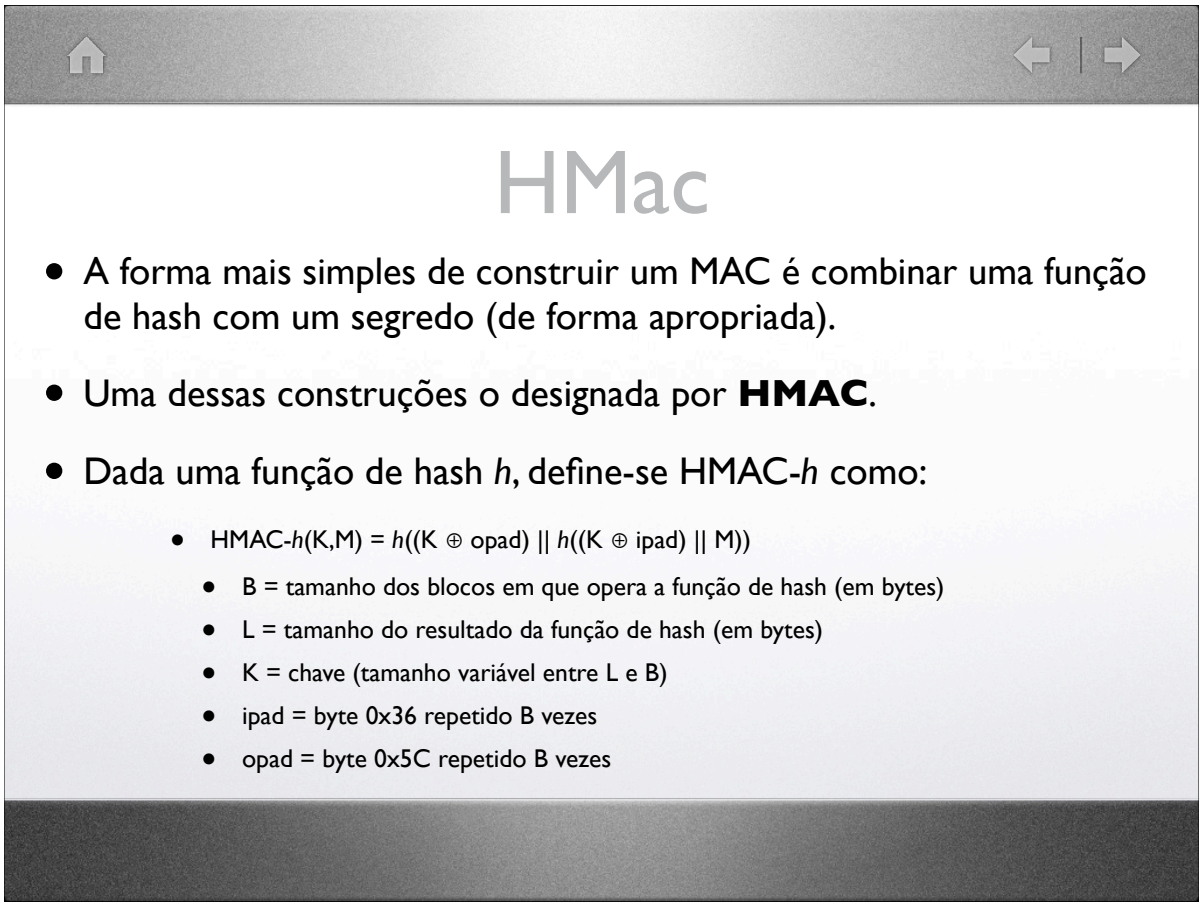


Figure 18.7 One SHA operation.

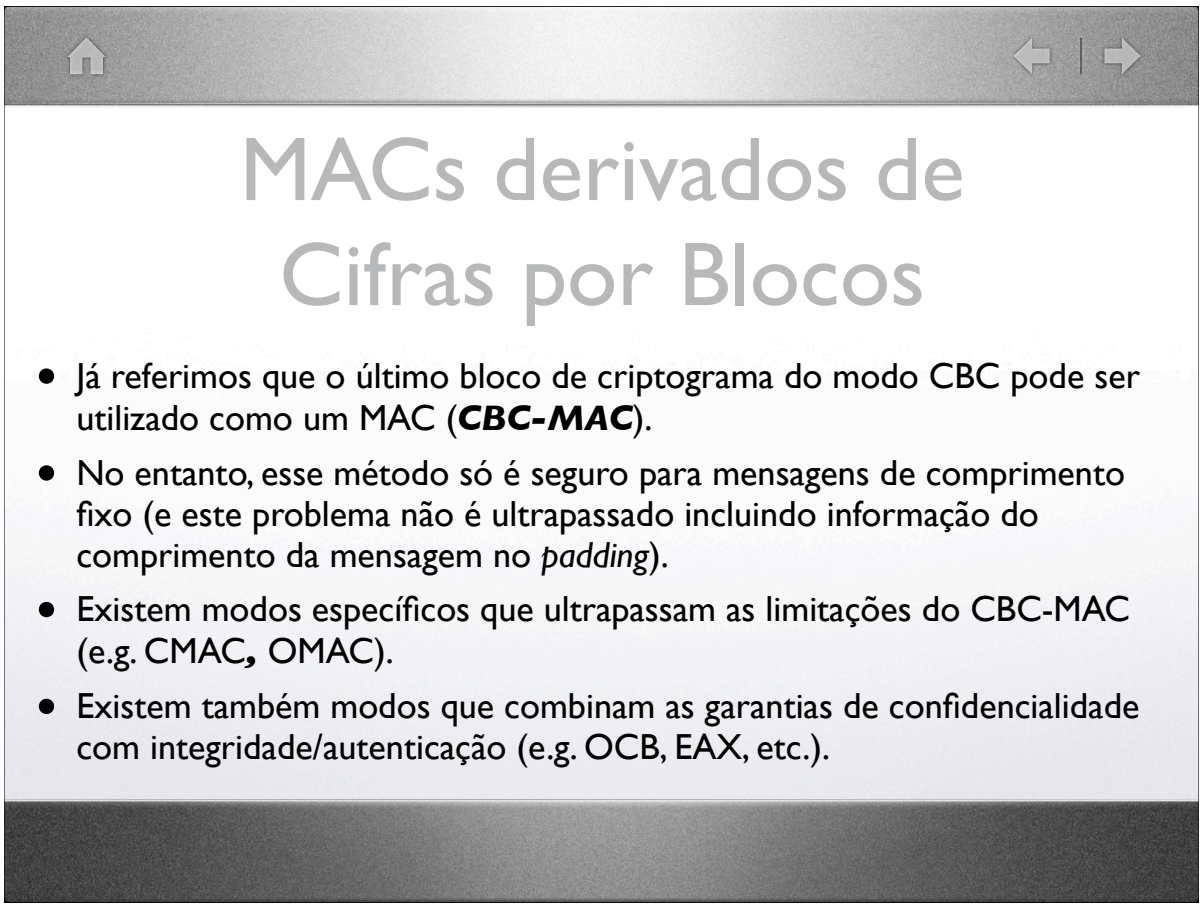
Message Authentication Codes (MAC)

- As funções de hash, por si só, não garantem nem a integridade nem autenticidade! (... mas quando utilizadas com uma cifra já permitem estabelecer essas propriedades)
- Um ***código de autenticação*** (MAC), pode ser entendido como "uma função de hash com segredo" e visa garantir essas propriedades.



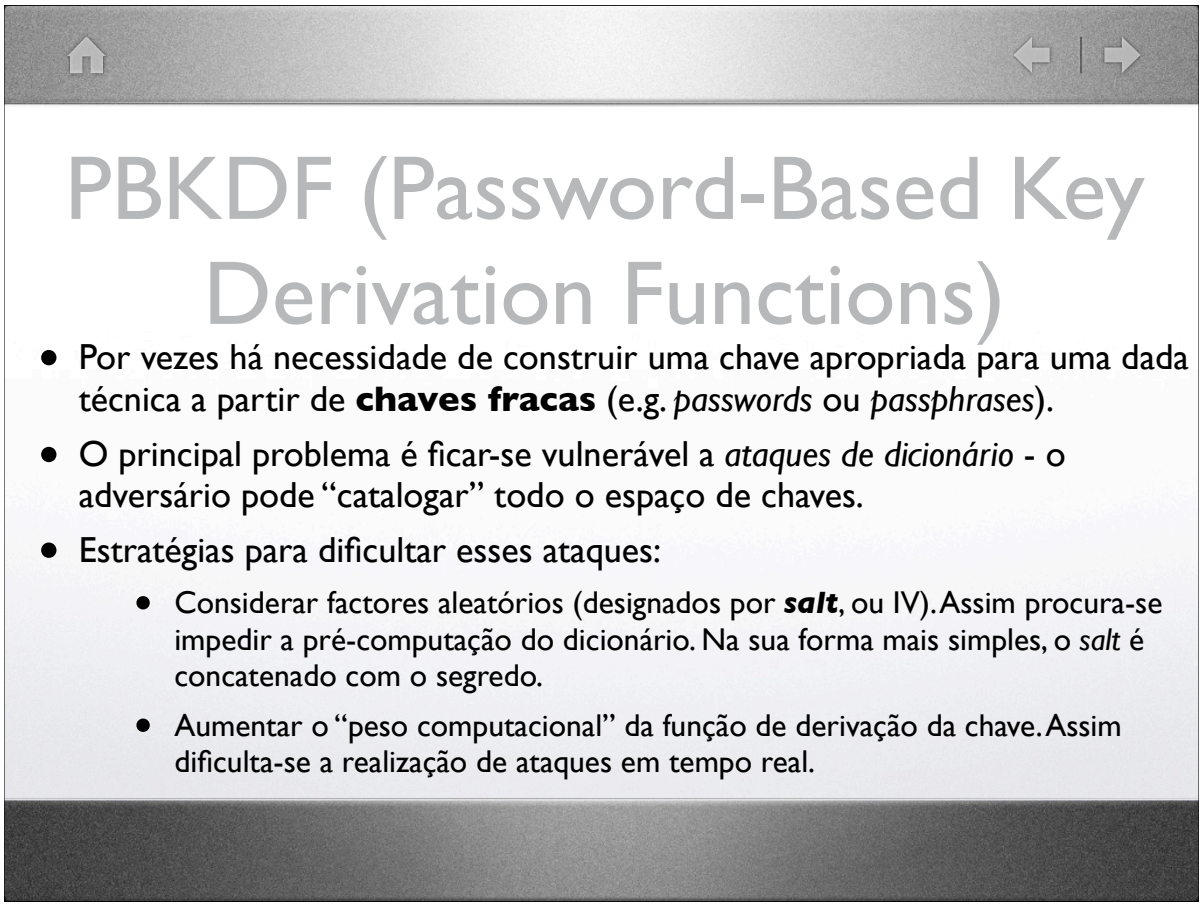
HMac

- A forma mais simples de construir um MAC é combinar uma função de hash com um segredo (de forma apropriada).
- Uma dessas construções o designada por **HMAC**.
- Dada uma função de hash *h*, define-se HMAC-*h* como:
  - HMAC-*h*(K,M) = *h*((K ⊕ opad) || *h*((K ⊕ ipad) || M))
    - B = tamanho dos blocos em que opera a função de hash (em bytes)
    - L = tamanho do resultado da função de hash (em bytes)
    - K = chave (tamanho variável entre L e B)
    - ipad = byte 0x36 repetido B vezes
    - opad = byte 0x5C repetido B vezes



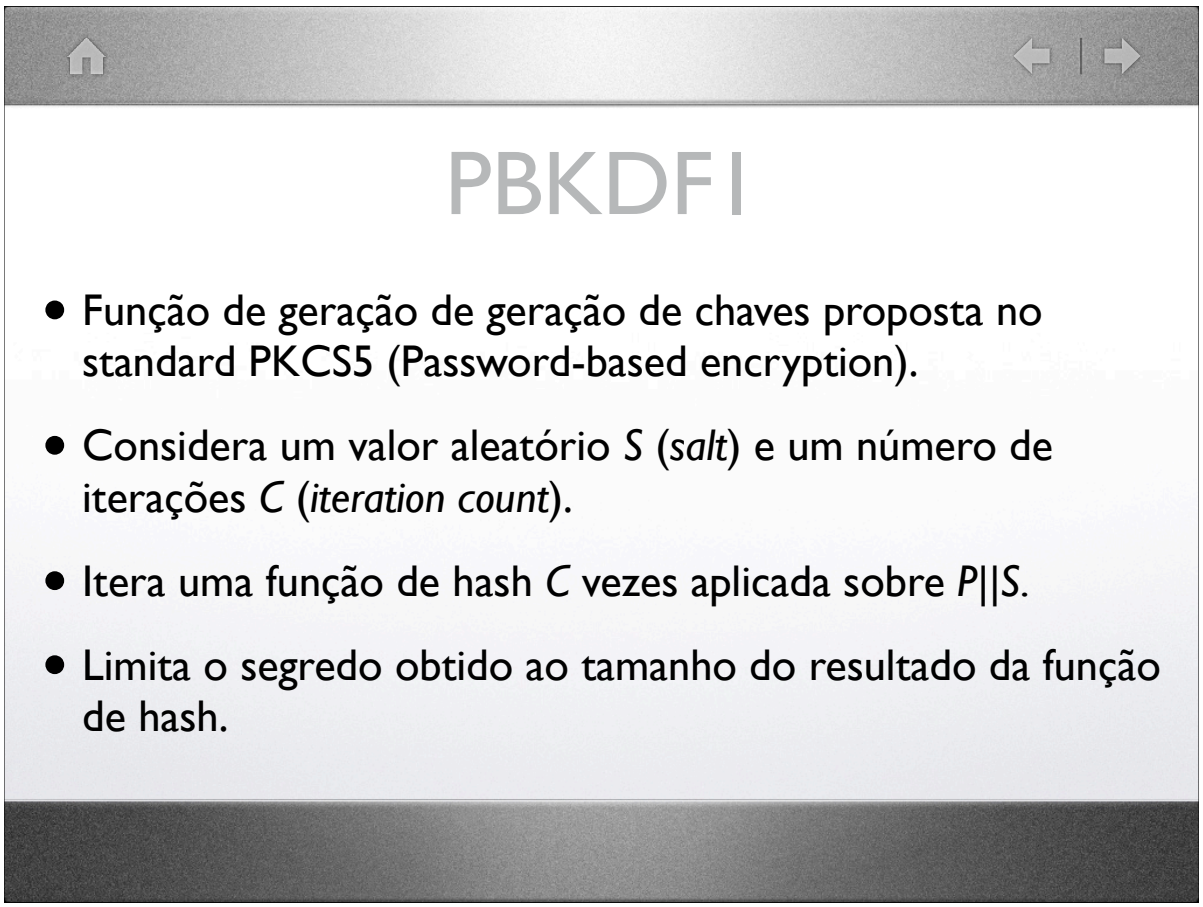
MACs derivados de Cifras por Blocos

- Já referimos que o último bloco de criptograma do modo CBC pode ser utilizado como um MAC (***CBC-MAC***).
- No entanto, esse método só é seguro para mensagens de comprimento fixo (e este problema não é ultrapassado incluindo informação do comprimento da mensagem no *padding*).
- Existem modos específicos que ultrapassam as limitações do CBC-MAC (e.g. CMAC, OMAC).
- Existem também modos que combinam as garantias de confidencialidade com integridade/autenticação (e.g. OCB, EAX, etc.).



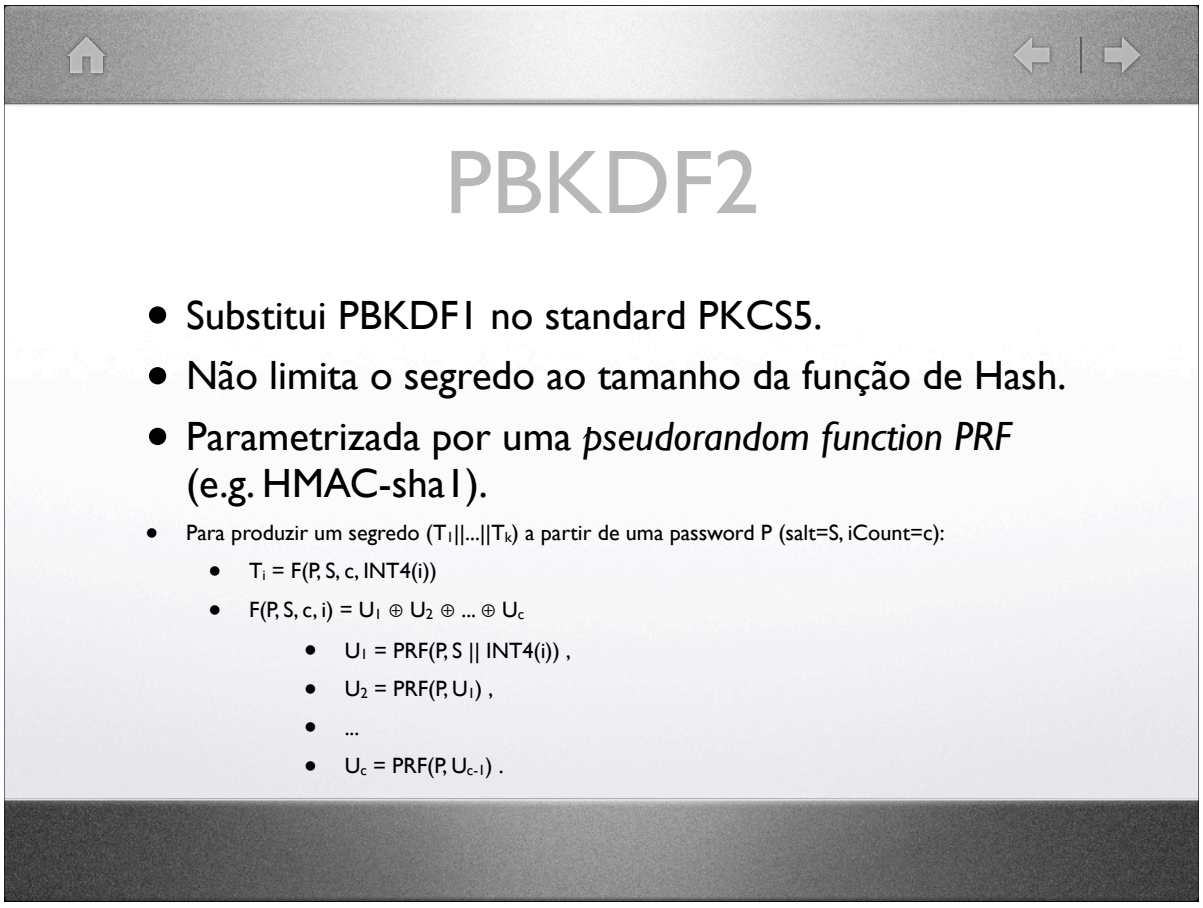
PBKDF (Password-Based Key Derivation Functions)

- Por vezes há necessidade de construir uma chave apropriada para uma dada técnica a partir de **chaves fracas** (e.g. *passwords* ou *passphrases*).
- O principal problema é ficar-se vulnerável a *ataques de dicionário* - o adversário pode "catalogar" todo o espaço de chaves.
- Estratégias para dificultar esses ataques:
  - Considerar factores aleatórios (designados por ***salt***, ou IV). Assim procura-se impedir a pré-computação do dicionário. Na sua forma mais simples, o *salt* é concatenado com o segredo.
  - Aumentar o "peso computacional" da função de derivação da chave. Assim dificulta-se a realização de ataques em tempo real.



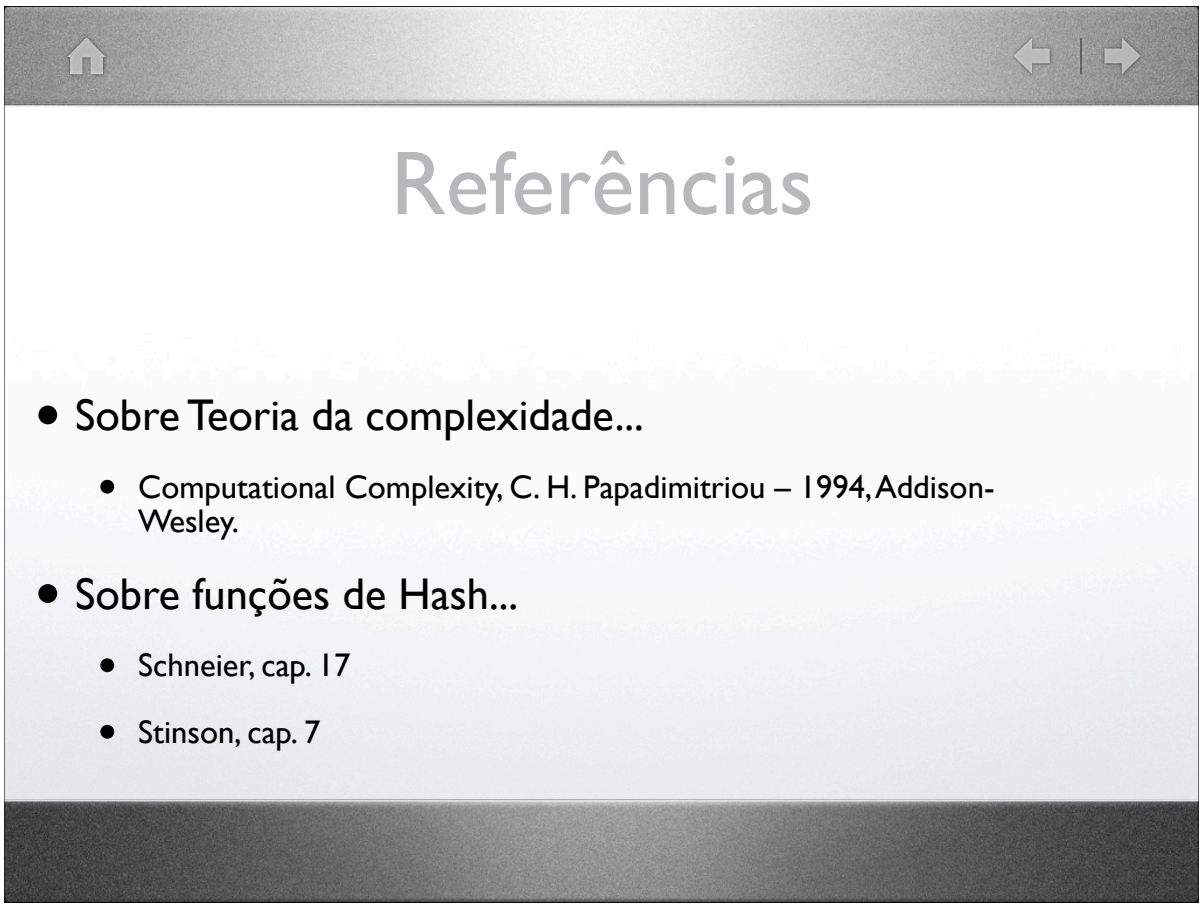
PBKDF1

- Função de geração de geração de chaves proposta no standard PKCS5 (Password-based encryption).
- Considera um valor aleatório *S* (*salt*) e um número de iterações *C* (*iteration count*).
- Itera uma função de hash *C* vezes aplicada sobre *P||S*.
- Limita o segredo obtido ao tamanho do resultado da função de hash.



PBKDF2

- Substitui PBKDF1 no standard PKCS5.
- Não limita o segredo ao tamanho da função de Hash.
- Parametrizada por uma *pseudorandom function PRF* (e.g. HMAC-sha1).
- Para produzir um segredo ($T_1||...||T_k$) a partir de uma password P (salt=S, iCount=c):
  - T_i = F(P, S, c, INT4(i))
  - F(P, S, c, i) = $U_1 \oplus U_2 \oplus ... \oplus U_c$
    - U_1 = PRF(P, S || INT4(i)) ,
    - U_2 = PRF(P, U_1) ,
    - ...
    - U_c = PRF(P, U_{c-1}) .



Referências

- Sobre Teoria da complexidade...
  - Computational Complexity, C. H. Papadimitriou – 1994, Addison-Wesley.
- Sobre funções de Hash...
  - Schneier, cap. 17
  - Stinson, cap. 7