Aula 9 - Cifras de bloco

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação
Universidade Federal do Rio de Janeiro
gabrielaquino@ic.ufrj.br

Compilado em:
September 19, 2025

Cifras de Fluxo vs Cifras de Bloco

Cifras de Fluxo:

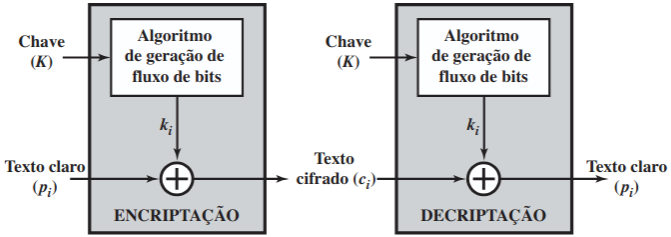
- Encriptam dados um **bit** ou **byte** por vez.
- Exemplos clássicos: Vigenère autochaveada e Vernam.
- Ideal: One-time pad (inquebrável se o fluxo de chaves for verdadeiramente aleatório).
- Limitação prática: o fluxo de chaves precisa ser compartilhado previamente via canal seguro.
- Solução prática: gerar o fluxo de bits algoritmicamente, controlado por chave, garantindo que partes futuras do fluxo sejam imprevisíveis.

Cifras de Bloco:

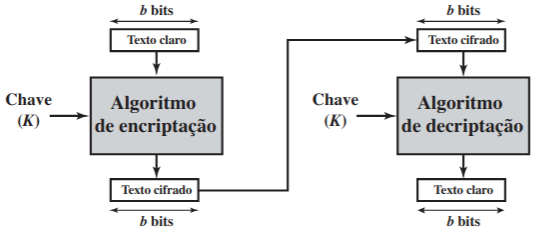
- Tratam um **bloco** de texto claro como um todo, normalmente de 64 ou 128 bits.
- Usuários compartilham uma chave simétrica.
- Modos de operação permitem uso semelhante ao das cifras de fluxo.
- Mais analisadas e amplamente utilizadas em aplicações de criptografia simétrica.

Cifra de fluxo vs. Cifra de bloco

Figura 3.1 Cifra de fluxo e cifra de bloco.



(a) Cifra de fluxo usando gerador algorítmico de fluxo de bits



Confusão e Difusão segundo Claude Shannon

Contexto histórico:

- Claude Shannon desenvolveu a ideia de **cifras de produto**, alternando funções de confusão e difusão [**?**].
- A estrutura da **cifra de Feistel**, baseada na proposta de Shannon de 1945, é utilizada em muitas cifras de bloco simétricas atuais.

Conceitos de Shannon:

- **Difusão**: espalhar a influência de cada bit do texto claro por muitos bits do texto cifrado, dificultando deduções estatísticas.
- **Confusão**: tornar a relação entre chave e texto cifrado complexa, de modo que conhecer parte da saída não revele informações sobre a chave.

Objetivo: impedir criptoanálise baseada em estatísticas do texto claro, como frequências de letras ou palavras prováveis.

Difusão em Criptografia

A difusão busca dissociar a estrutura estatística do texto claro das estatísticas do texto cifrado.

Princípio: Cada dígito do texto claro deve afetar muitos dígitos do texto cifrado, espalhando a informação.

Exemplo matemático: Para uma mensagem $M = m_1, m_2, m_3, \ldots$:

$$y_n = \left(\sum_{i=1}^{k} m_{n+i} \right) \bmod 26$$

onde k letras sucessivas do texto claro influenciam cada letra do texto cifrado.

Efeito:

- Frequências de letras e dígrafos no texto cifrado se aproximam de uma distribuição uniforme.
- Em cifras de bloco binárias, difusão é alcançada por permutações seguidas de funções de transformação, garantindo que bits de diferentes posições contribuam para cada bit cifrado.

Difusão e Confusão em Cifras de Bloco

Cada cifra de bloco transforma um bloco de texto claro em texto cifrado, dependendo da chave.

Difusão:

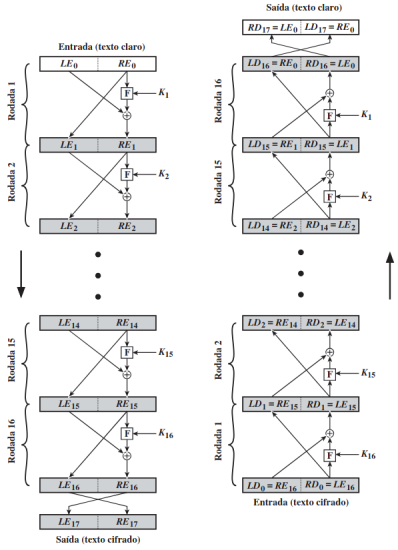
- Complica o relacionamento estatístico entre o texto claro e o texto cifrado.
- Faz com que cada bit do texto claro influencie muitos bits do texto cifrado.
- Frustra tentativas de deduzir padrões no texto claro.

Confusão:

- Complica o relacionamento entre o texto cifrado e a chave de encriptação.
- Utiliza funções de substituição complexas para dificultar a descoberta da chave.
- Funções lineares simples resultariam em pouca confusão e vulnerabilidade.

Cifra de Feistel

Figura 3.3 Encriptação e decriptação de Feistel (16 rodadas).



Estrutura da Cifra de Feistel

A estrutura de Feistel permite criar cifras de bloco seguras a partir de funções de encriptação simples.

Descrição:

- Entrada: bloco de texto claro de 2w bits e chave K.
- O bloco é dividido em duas metades: L_0 e R_0.
- Os dados passam por n rodadas de processamento.
- Cada rodada i recebe L_{i-1}, R_{i-1} e uma subchave K_i derivada de K.
- As subchaves K_i são normalmente diferentes entre si e da chave original.
- Após n rodadas, as metades são combinadas para gerar o bloco cifrado.

Exemplo: 16 rodadas, mas qualquer número de rodadas pode ser implementado.

Rodadas da Cifra de Feistel

Todas as rodadas têm a mesma estrutura básica:

Processamento em cada rodada:

1. Aplica-se uma função de substituição F à metade direita dos dados R_{i-1}, parametrizada pela subchave K_i.
2. Realiza-se a operação XOR entre a saída de F e a metade esquerda L_{i-1}:

$$L_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

3. Executa-se a permutação trocando as metades:

$$R_i = R_{i-1}, \quad L_i \leftrightarrow R_i$$

Observações:

- F é uma função de w bits de R_{i-1} e y bits de K_i, produzindo w bits de saída.
- Estrutura geral é repetida em todas as rodadas, formando uma rede de substituição-permutação (SPN) como proposta por Shannon.

Fatores que influenciam a segurança da cifra de Feistel

A execução de uma rede de Feistel depende de diversos parâmetros de projeto, sendo um dos principais:

Tamanho de bloco:

- Blocos maiores proporcionam maior segurança, mantendo os demais fatores constantes.
- Blocos maiores reduzem a velocidade de encriptação/decriptação para um dado algoritmo.
- Maior segurança é obtida por meio de maior difusão.
- Tradicionalmente, o tamanho de bloco de 64 bits era considerado adequado para cifras de bloco.
- O AES moderno utiliza tamanho de bloco de 128 bits.

Fatores que influenciam a segurança da cifra de Feistel

- **Tamanho da chave:**
  - Chave maior $\Rightarrow$ maior resistência a ataques de força bruta e maior confusão.
  - Impacto: pode reduzir a velocidade de encriptação/decriptação.
  - 64 bits ou menos: inseguros.
  - 128 bits: tornou-se padrão comum.
- **Número de rodadas:**
  - 1 rodada: segurança inadequada.
  - 16 rodadas: valor típico que garante segurança aceitável.
- **Algoritmo de geração de subchaves:**
  - Quanto mais complexo, mais difícil a criptoanálise.
- **Função F:**
  - Função mais complexa $\Rightarrow$ maior resistência a ataques.

Considerações adicionais no projeto da cifra de Feistel

- **Encriptação/Decriptação rápidas em software:**
  - Muitas vezes a implementação ocorre em software, não em hardware.
  - Desempenho do algoritmo passa a ser um fator crítico.
- **Facilidade de análise:**
  - Algoritmos claros e concisos são mais fáceis de avaliar contra ataques.
  - Transparência aumenta a confiança na robustez criptográfica.
  - Exemplo: o DES não possui estrutura de fácil análise.

Algoritmo de Decriptação de Feistel

O processo de decriptação com uma cifra de Feistel é basicamente o mesmo da encriptação. A regra é a seguinte: use o texto cifrado como entrada para o algoritmo, mas aplique as subchaves K_i em ordem reversa.

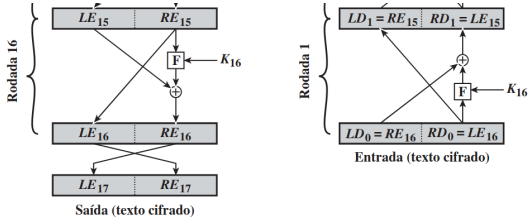
Ou seja:

- Use K_n na primeira rodada, K_{n-1} na segunda, ..., até K_1 na última rodada.

Observação: Isso permite que o mesmo algoritmo seja usado tanto para encriptação quanto para decriptação.

Validação do Algoritmo de Feistel com Ordem de Chaves Invertida

Observando a última rodada de encriptação da Cifra de Feistel e a primeira rodada da decriptação

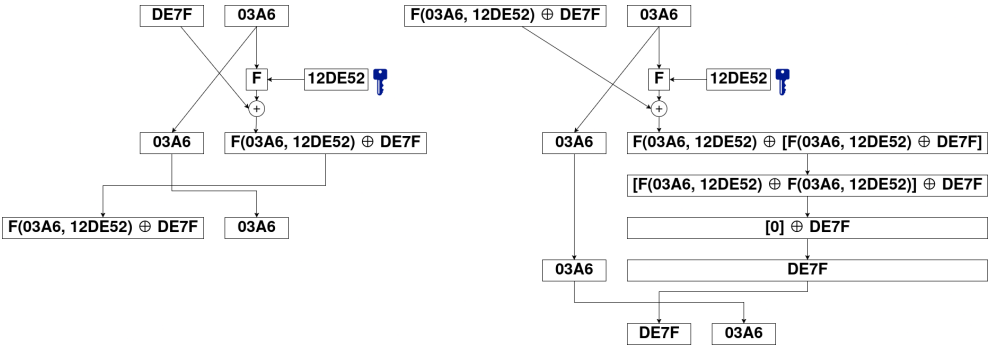


E sabendo que a operação lógica ou-exclusivo tem as seguintes propriedades:

$$[A \oplus B] \oplus C = A \oplus [B \oplus C]$$
$$D \oplus D = 0$$
$$E \oplus 0 = E$$

Validação do Algoritmo de Feistel com Ordem de Chaves Invertida

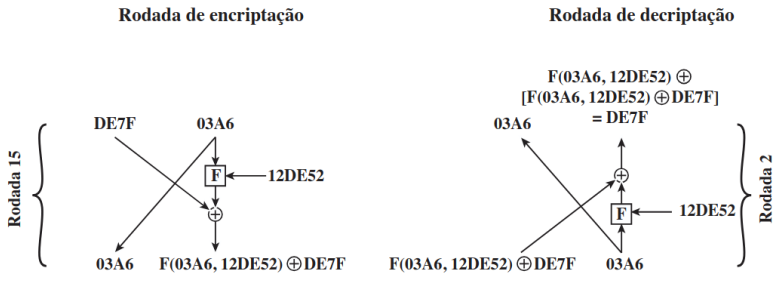
Temos:



Resumindo

Temos:

Figura 3.4 Exemplo Feistel.



Data Encryption Standard (DES)

Até a introdução do **Advanced Encryption Standard (AES)** em 2001, o DES era o esquema de encriptação mais utilizado.

Histórico:

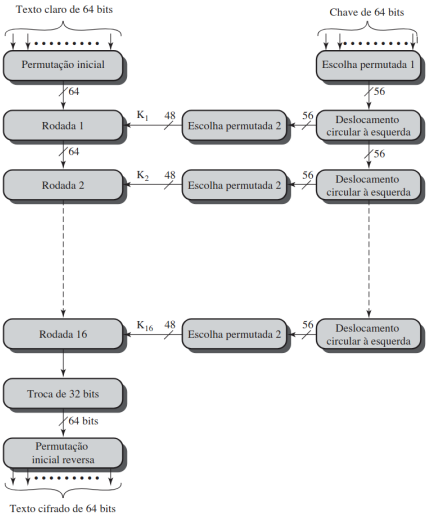
- Adotado em 1977 pelo **NIST**.
- Também conhecido como **Data Encryption Algorithm (DEA)**.
- Encripta dados em blocos de 64 bits usando uma chave de 56 bits.
- O mesmo algoritmo e chave são usados para encriptação e decriptação.

Evolução:

- Em 1994, o NIST reafirmou o DES para uso federal, recomendando restrição a informações não confidenciais.
- Em 1999, o NIST indicou que o DES seria apenas para sistemas legados e recomendou o **Triple DES**.
  - Triple DES repete o algoritmo DES três vezes usando duas ou três chaves diferentes.
- Hoje recomendamos o uso do **AES**

Esquema do DES

Figura 3.5 Representação geral do algoritmo de encriptação DES.



Funcionamento DES

No esquema de encriptação DES existem duas entradas na função: **texto claro** (64 bits) e **chave** (56 bits).

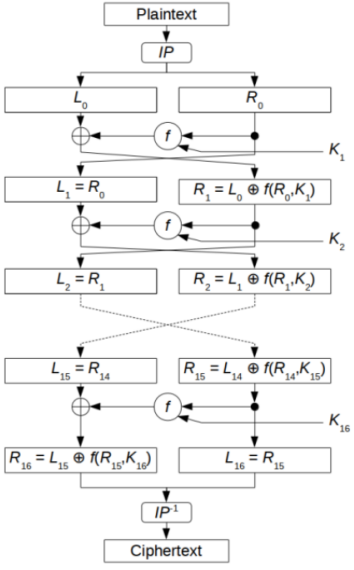
Processamento do texto claro:

- **Permutação Inicial (IP)**: reorganiza os bits do texto claro para produzir a entrada permutada.
- **16 rodadas de função Feistel**: cada rodada envolve permutação e substituição dos bits.
- **Troca das metades esquerda e direita** para gerar a pré-saída.
- **Permutação Final (IP^{-1})**: inverso da permutação inicial, produzindo o texto cifrado de 64 bits.

Data Encryption Standard (DES)

- Cifra de bloco que processa dados em blocos de 64 bits.
- Entrada: bloco de texto claro de 64 bits.
- Saída: bloco de texto cifrado de 64 bits.
- Algoritmo simétrico: mesma chave e mesmo algoritmo para encriptação e decriptação (com pequenas diferenças no escalonamento de chaves).

Esquema detalhado - DES



Chaves no DES

- Comprimento efetivo da chave: 56 bits.
- A chave é representada como um número de 64 bits:
  - A cada byte, o bit menos significativo é usado para verificação de paridade.
  - Esses 8 bits de paridade não participam da criptografia.
- Existem algumas chaves fracas, mas são facilmente evitáveis.
- Toda a segurança do DES depende da chave utilizada.

Funcionamento DES

Processamento da chave:

- Chave de 56 bits passa por uma permutação inicial.
- Para cada uma das 16 rodadas, gera-se uma subchave (K_i) usando:
  - Deslocamento circular à esquerda.
  - Permutação fixa.
- Cada rodada utiliza uma subchave diferente, derivada da chave original.

Decriptação DES: Assim como qualquer cifra de Feistel, a decriptação usa o mesmo algoritmo da encriptação, exceto que a aplicação das subchaves é invertida. Além disso, as permutações inicial e final são invertidas.

Outline of the DES Algorithm

- O DES opera sobre blocos de **64 bits**.
- Etapas principais:
  - **Permutação inicial (IP)** sobre o bloco de entrada.
  - Divisão em duas metades:
    - Lado esquerdo (32 bits).
    - Lado direito (32 bits).

Rodadas e Finalização do DES

- São realizadas **16 rodadas** de operações idênticas.
- Em cada rodada:
  - A função f combina dados com subchaves derivadas da chave principal.
- Após a 16ª rodada:
  - As metades esquerda e direita são reunidas.
  - Aplica-se a **permutação final**, inversa da permutação inicial.

DES Round Function f

- Em cada rodada:
  - A chave de 56 bits é **deslocada** e **reduzida** para 48 bits.
  - O lado direito (32 bits) é **expandido** para 48 bits.
  - Os 48 bits resultantes são combinados com a subchave via **XOR**.
  - O resultado passa por **8 S-boxes**, produzindo 32 bits.
  - Esses 32 bits sofrem uma **permutação**.
- A saída da função f é combinada com o lado esquerdo via XOR.
- Após isso, ocorre a **troca de metades**.

Equações de uma Rodada no DES

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

- L_i, R_i: metades esquerda e direita no final da rodada i.
- K_i: subchave de 48 bits da rodada i.
- f: função que realiza expansão, XOR, substituições e permutação.
- Repetido por **16 rodadas**.

Uma rodada do DES

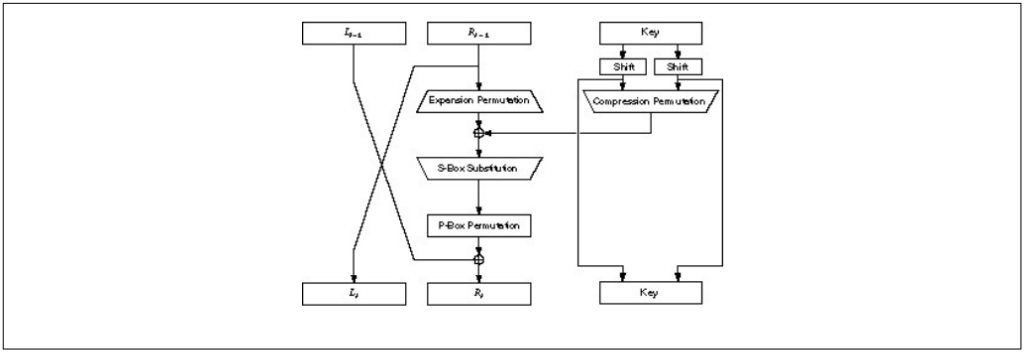


Figure 12.2 *One round of DES.*

Exemplo do DES

Exemplo prático do DES:

- **Objetivo**: observar os padrões em hexadecimal que surgem de uma etapa para outra, sem necessariamente replicar o cálculo manualmente
- Texto claro escolhido é um **palíndromo hexadecimal**.

Texto claro:	`02468aceeca86420`
Chave:	`0f1571c947d9e859`
Texto cifrado:	`da02ce3a89ecac3b`

Exemplo do DES

- Evolução do algoritmo ao longo das rodadas.
- Primeira linha: valores de 32 bits das metades esquerda (L) e direita (R) **após a permutação inicial (IP)**.
- Linhas seguintes (1 a 16): resultado após cada rodada, incluindo:
- Última linha: valores de L e R após a permutação inicial inversa (IP^{-1}).
- A concatenação desses dois valores gera o **texto cifrado**.

Tabela 3.2 Exemplo de DES.

Rodada	K_i	L_i	R_i
IP		5a005a00	3cf03c0f
1	1e030f03080d2930	3cf03c0f	bad22845
2	0a31293432242318	bad22845	99e9b723
3	23072318201d0c1d	99e9b723	0bae3b9e
4	05261d3824311a20	0bae3b9e	42415649
5	3325340136002c25	42415649	18b3fa41
6	123a2d0d04262a1c	18b3fa41	9616fe23
7	021f120b1c130611	9616fe23	67117cf2
8	1c10372a2832002b	67117cf2	c11bfc09
9	04292a380c341f03	c11bfc09	887fbc6c
10	2703212607280403	887fbc6c	600f7e8b
11	2826390c31261504	600f7e8b	f596506e
12	12071c241a0a0f08	f596506e	738538b8
13	300935393c0d100b	738538b8	c6a62c4e
14	311e09231321182a	c6a62c4e	56b0bd75
15	283d3e0227072528	56b0bd75	75e8fd8f
16	2921080b13143025	75e8fd8f	25896490
IP^{-1}		da02ce3a	89ecac3b

Nota: as subchaves DES são apresentadas como oito valores de 6 bits no padrão hexa.

Efeito Avalanche

- Propriedade desejável em algoritmos de encriptação.
- Uma pequena mudança no texto claro ou na chave deve causar uma grande alteração no texto cifrado.
- **Objetivo**: Alterar apenas **um bit** no texto claro ou na chave deve modificar **muitos bits** no resultado.
- **Resultado esperado**: Caso a mudança fosse pequena, seria possível reduzir o espaço de busca de textos claros ou de chaves.
- O efeito avalanche aumenta a segurança, dificultando ataques de análise e padrões previsíveis.

Exemplo do Efeito Avalanche no DES

- Texto claro inicial modificado no 4º bit.
- Tabela de acompanhamento mostra:
  - Coluna 1: rodadas do algoritmo.
  - Coluna 2: valores intermediários de 64 bits no final de cada rodada.
  - Coluna 3: número de bits diferentes entre os dois textos.
- Após apenas 3 rodadas: diferença de **18 bits**.
- Texto cifrado final: diferença de **32 bits**.
- Demonstra claramente o **efeito avalanche** do DES.

Exemplo do Efeito Avalanche no DES

Tabela 3.3 Efeito avalanche no DES: mudança no texto claro.

Rodada		δ
	02468aceeca86420 12468aceeca86420	1
1	3cf03c0fbad22845 3cf03c0fbad32845	1
2	bad2284599e9b723 bad3284539a9b7a3	5
3	99e9b7230bae3b9e 39a9b7a3171cb8b3	18
4	0bae3b9e42415649 171cb8b3ccaca55e	34
5	4241564918b3fa41 ccaca55ed16c3653	37
6	18b3fa419616fe23 d16c3653cf402c68	33
7	9616fe2367117cf2 cf402c682b2cefbc	32
8	67117cf2c11bfc09 2b2cefbc99f91153	33
9	c11bfc09887fbc6c 99f911532eed7d94	32
10	887fbc6c600f7e8b 2eed7d94d0f23094	34
11	600f7e8bf596506e d0f23094455da9c4	37
12	f596506e738538b8 455da9c47f6e3cf3	31
13	738538b8c6a62c4e 7f6e3cf34bc1a8d9	29
14	c6a62c4e56b0bd75 4bc1a8d91e07d409	33
15	56b0bd7575e8fd8f 1e07d4091ce2e6dc	31
16	75e8fd8f25896490 1ce2e6dc365e5f59	32
IP^{-1}	da02ce3a89ecac3b 057cde97d7683f2a	32

Efeito Avalanche com Alteração da Chave

Experimento: Texto claro mantido constante mas com duas chaves utilizadas:

- Chave original modificada **com diferença no 4º bit**

Resultados:

- Diferença significativa no texto cifrado final — cerca de metade dos bits são diferentes.
- O efeito avalanche já aparece após poucas rodadas do DES.

Demonstra: Robustez do algoritmo contra pequenas alterações na chave.

Exemplo do Efeito Avalanche no DES - Mudança na chave

Tabela 3.4 Efeito avalanche no DES: mudança na chave.

Rodada		δ
	02468aceeca86420 02468aceeca86420	0
1	3cf03c0fbad22845 3cf03c0f9ad628c5	3
2	bad2284599e9b723 9ad628c59939136b	11
3	99e9b7230bae3b9e 9939136b768067b7	25
4	0bae3b9e42415649 768067b75a8807c5	29
5	4241564918b3fa41 5a8807c5488dbe94	26
6	18b3fa419616fe23 488dbe94aba7fe53	26
7	9616fe2367117cf2 aba7fe53177d21e4	27
8	67117cf2c11bfc09 177d21e4548f1de4	32
9	c11bfc09887fbc6c 548f1de471f64dfd	34
10	887fbc6c600f7e8b 71f64dfd4279876c	36
11	600f7e8bf596506e 4279876c399fdc0d	32
12	f596506e738538b8 399fdc0d6d208dbb	28
13	738538b8c6a62c4e 6d208dbbb9bdeeaa	33
14	c6a62c4e56b0bd75 b9bdeeaad2c3a56f	30
15	56b0bd7575e8fd8f d2c3a56f2765c1fb	33
16	75e8fd8f25896490 2765c1fb01263dc4	30
IP^{-1}	da02ce3a89ecac3b ee92b50606b62b0b	30

A Força do DES

- Desde sua adoção como padrão federal, surgiram preocupações sobre a segurança do DES.
- Principais áreas de preocupação:
  - **Tamanho da chave**: 56 bits pode ser vulnerável a ataques de força bruta.
  - **Natureza do algoritmo**: estrutura e procedimentos internos do DES podem influenciar sua resistência a ataques criptográficos.
- Essas questões motivaram o desenvolvimento de algoritmos mais seguros, como o Triple DES (3DES) e o AES.

Uso de Chaves de 56 Bits no DES

- Chave de 56 bits $\rightarrow 2^{56} \approx 7,2 \times 10^{16}$ chaves possíveis.
- Ataque de força bruta **pareceria** impraticável:
  - Pesquisando metade do espaço de chaves
  - Uma máquina realizando 1 encriptação por microssegundo levaria mais de 1000 anos.
- No entanto, Diffie e Hellman (1977) propuseram tecnologia paralela:
  - Máquina com 1 milhão de dispositivos, cada um realizando 1 encriptação/µs.
  - Tempo médio de busca: cerca de 10 horas.
  - Custo estimado: US$ 20 milhões (em 1977).
- Demonstra que, apesar do tamanho da chave, o DES não é invulnerável a ataques especializados.

Impacto da Tecnologia Moderna na Segurança do DES

- Atualmente, nem é necessário hardware especial para ataques de força bruta contra o DES.
- Processadores comerciais modernos já oferecem velocidades que ameaçam a segurança do DES:
  - Computadores multicore atuais: 10^9 combinações de chaves por segundo [SEAG08].
  - Testes em máquinas Intel multicore: 5×10^8 encriptações por segundo [BASU12].
  - Supercomputadores contemporâneos: 10^{13} encriptações por segundo.
- Instruções de hardware recentes para AES também aceleram operações criptográficas, mostrando que ataques de força bruta se tornam cada vez mais viáveis.
- Conclusão: o DES, com chave de 56 bits, é vulnerável diante da tecnologia atual.

Força Bruta e Tamanho da Chave

- Tabela 3.5 mostra o tempo necessário para ataques de força bruta em diferentes tamanhos de chave.
- Exemplos:
  - Um único PC moderno pode quebrar o DES (56 bits) em 1 ano.
  - Vários PCs em paralelo reduzem drasticamente o tempo.
  - Supercomputadores contemporâneos: 1 hora para descobrir a chave do DES.
- Chaves de 128 bits ou mais são efetivamente inquebráveis por força bruta:
  - Mesmo com aceleração de 10^{12} vezes, ainda seriam necessários 100 mil anos.
- Alternativas mais seguras ao DES: **AES** e **Triple DES**

Exemplo do Efeito Avalanche no DES - Mudança na chave

Tabela 3.5 Tempo médio exigido para uma busca exaustiva no espaço de chaves.

Tamanho de chave (bits)	Cifra	Número de chaves alternativas	Tempo exigido a 10^9 decriptações/s	Tempo exigido a 10^{13} decriptações/s
56	DES	$2^{56} \approx 7{,}2 \times 10^{16}$	2^{55} ns = 1,125 ano	1 hora
128	AES	$2^{128} \approx 3{,}4 \times 10^{38}$	2^{127} ns = $5{,}3 \times 10^{21}$ anos	$5{,}3 \times 10^{17}$ anos
168	Triple DES	$2^{168} \approx 3{,}7 \times 10^{50}$	2^{167} ns = $5{,}8 \times 10^{33}$ anos	$5{,}8 \times 10^{29}$ anos
192	AES	$2^{192} \approx 6{,}3 \times 10^{57}$	2^{191} ns = $9{,}8 \times 10^{40}$ anos	$9{,}8 \times 10^{36}$ anos
256	AES	$2^{256} \approx 1{,}2 \times 10^{77}$	2^{255} ns = $1{,}8 \times 10^{60}$ anos	$1{,}8 \times 10^{56}$ ano
26 caracteres (permutação)	Monoalfabético	$2! = 4 \times 10^{26}$	2×10^{26} ns = $6{,}3 \times 10^{9}$ anos	$6{,}3 \times 10^{6}$ anos

Resumo - Projeto de cifra de bloco

Número de Rodadas no DES

- Maior número de rodadas $\rightarrow$ mais difícil a criptoanálise, mesmo se a função F for relativamente fraca.
- Critério de projeto: número de rodadas suficiente para que criptoanálises conhecidas exijam mais esforço do que um ataque de força bruta.
- No DES:
  - 16 rodadas
  - Criptoanálise diferencial: $2^{55,1}$ operações
  - Força bruta: 2^{55} operações
  - Se houvesse 15 ou menos rodadas, criptoanálise diferencial exigiria menos esforço que a força bruta.
- Critério facilita comparar a força de diferentes algoritmos.
- Sem descobertas revolucionárias em criptoanálise, a força de um algoritmo que satisfaz este critério é julgada principalmente pelo tamanho da chave.

Resumo - Projeto de cifra de bloco

Projeto da Função F no DES

- A função F é o núcleo da cifra de bloco de Feistel e é responsável pela **confusão**.
- Critérios importantes para o projeto de F:
  - **Não linearidade**: quanto menos linear, mais difícil a criptoanálise.
  - **Efeito avalanche**: mudança em 1 bit da entrada altera muitos bits da saída.
  - **Strict Avalanche Criterion (SAC)** [WEBS86]:
    - Qualquer bit de saída muda com probabilidade $1/2$ quando qualquer bit de entrada é invertido.
    - Originalmente definido para S-boxes, mas aplicável à função F como um todo.
  - **Bit Independence Criterion (BIC)** [WEBS86]:
    - Bits de saída mudam independentemente quando qualquer bit de entrada é invertido.
- Aplicar SAC e BIC fortalece a eficácia da função de confusão e aumenta a segurança do algoritmo.

Resumo - Projeto de cifra de bloco

Algoritmo de Escalonamento de Chave

- Em cifras de bloco de Feistel, a chave principal é usada para gerar uma **subchave** para cada rodada.
- Objetivo do escalonamento de chave:
  - Maximizar a dificuldade de deduzir subchaves individuais.
  - Tornar mais difícil recuperar a chave principal a partir das subchaves.
- Não existe, até o momento, um princípio geral universalmente aceito para o projeto do algoritmo de escalonamento de chaves.
- O cuidado no escalonamento é crucial para garantir a segurança global do algoritmo.

Triple DES (3DES)

- Primeiro padronizado em 1985 pela ANSI para aplicações financeiras (X9.17).
- Incorporado ao DES como parte do padrão FIPS 46-3 em 1999.
- Utiliza **três chaves** e três execuções do algoritmo DES.
- Sequência de operação: **Encrypt-Decrypt-Encrypt (EDE)**:

$$C = E(K_3, D(K_2, E(K_1, P)))$$

  - C = texto cifrado
  - P = texto claro
  - K_1, K_2, K_3 = chaves
- A abordagem EDE garante compatibilidade com DES original quando $K_1 = K_2 = K_3$.

Diretrizes do FIPS 46-3 para 3DES

- 3DES é o algoritmo de cifra simétrica aprovado pelo FIPS para uso corrente.
- O DES original (chave de 56 bits) é permitido apenas para sistemas legados.
- Novas aquisições de sistemas devem suportar 3DES.
- Organizações governamentais com sistemas DES legados são incentivadas a migrar para 3DES.
- Espera-se que 3DES e AES coexistam como algoritmos aprovados pelo FIPS, permitindo uma transição gradual para AES.

Exemplo do Efeito Avalanche no DES - Mudança na chave

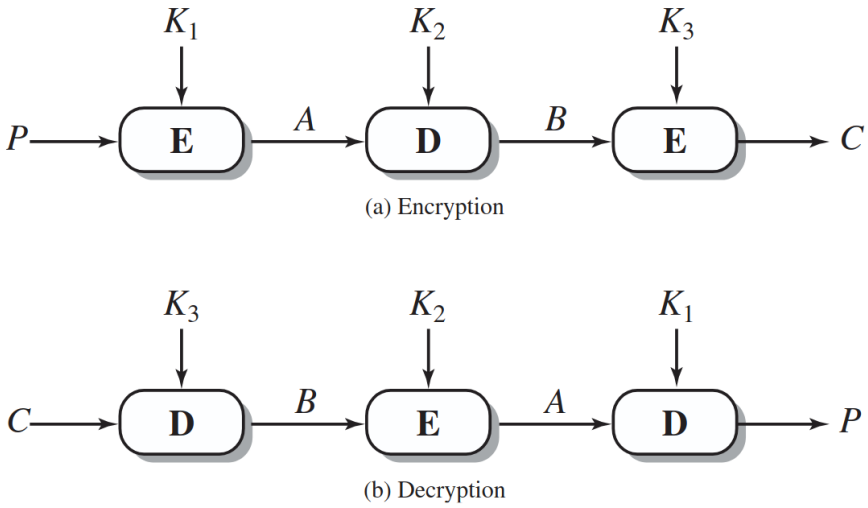


Figure 20.2 **Triple DES**

Compatibilidade do 3DES com DES

- Quando as três chaves são iguais ($K_1 = K_2 = K_3$), temos:

$$C = E(K_1, D(K_1, E(K_1, P))) = E(K, P)$$

- A operação do meio $D(K_1, E(K_1, P))$ se anula, resultando apenas na cifra DES original.
- Importância:
  - Permite que dados cifrados com DES possam ser decifrados por um sistema 3DES configurado em modo compatível.
  - Sistemas antigos que usam DES podem processar dados cifrados por 3DES no modo compatível.
- Essa retrocompatibilidade facilitou a adoção do 3DES sem necessidade de substituir todos os sistemas legados.

Chaves e Criptografia no 3DES

- O uso da decriptação na segunda etapa do 3DES não tem significado criptográfico; sua vantagem é permitir compatibilidade com o DES original:

$$C = E(K_1, D(K_1, E(K_1, P))) = E(K, P)$$

- Com três chaves distintas (K_1, K_2, K_3), o 3DES possui **168 bits efetivos de chave**.
- É possível usar apenas duas chaves $(K_1 = K_3)$, resultando em uma chave efetiva de **112 bits**.
- O padrão FIPS 46-3 fornece diretrizes formais para a utilização do 3DES com estas configurações.

Chaves e Criptografia no 3DES

- O uso da decriptação na segunda etapa do 3DES não tem significado criptográfico; sua vantagem é permitir compatibilidade com o DES original:

$$C = E(K_1, D(K_1, E(K_1, P))) = E(K, P)$$

- Com três chaves distintas (K_1, K_2, K_3), o 3DES possui **168 bits efetivos de chave**.
- É possível usar apenas duas chaves $(K_1 = K_3)$, resultando em uma chave efetiva de **112 bits**.
- O padrão FIPS 46-3 fornece diretrizes formais para a utilização do 3DES com estas configurações.

Advanced Encryption Standard (AES) e Rijndael

- Publicado pelo **NIST** em 2001 como padrão de cifra simétrica de bloco.
- Substitui o DES em diversas aplicações, oferecendo maior segurança e eficiência.
- AES é mais complexo que cifras de chave pública (ex.: RSA), sendo difícil de explicar de forma simplificada.
- Em 2000, o NIST selecionou a família de cifras de bloco **Rijndael** como vencedora do concurso AES.
- **Rijndael**: cifra de bloco escolhida como AES pelo NIST.
- **AES não usa Cifra de Feistel**
- Três variantes especificadas: AES-128; AES-192; AES-256

Table 3. Key-Block-Round Combinations

	Key length		Block size		Number of rounds
	Nk	(in bits)	*Nb*	(in bits)	*Nr*
AES-128	4	128	4	128	10
AES-192	6	192	4	128	12
AES-256	8	256	4	128	14

Diferenças entre AES-128, AES-192 e AES-256

- As três variantes diferem em três aspectos principais:
  1. **Comprimento da chave** (128, 192 ou 256 bits).
  2. **Número de rodadas (Nr)**, que determina o tamanho do key schedule.
  3. **Especificação da recursão em KEY EXPANSION()**.
- Para cada algoritmo:
  - **Nr**: número de rodadas.
  - **Nk**: número de palavras da chave.
  - **Nb**: número de palavras do estado; no padrão AES, **Nb = 4**.
- Somente essas configurações de Rijndael estão conformes com o padrão AES.
- Veja a especificação FIPS 197 do AES

AES também suporta vários modos

Exemplos de modos de operação:

- AES-ECB (Electronic Codebook)
- AES-CBC (Cipher Block Chaining)
- AES-CTR (Counter Mode)
- AES-GCM (Galois/Counter Mode)
- Artigo: A Comparative Analysis of AES Common Modes of Operation
- Artigo: MODES OF OPERATION OF THE AES ALGORITHM

Electronic Code Book (ECB) – Considerações

- O modo ECB divide a mensagem em blocos $(P_1, P_2, \ldots, P_N)$ e cifra cada bloco separadamente com a mesma chave K.
- Se a mensagem não preencher o último bloco, ele é completado com *padding*.
- Vantagem: erros em um bloco não se propagam para os outros, permitindo decifrar blocos não corrompidos.
- Desvantagem: a criptografia é determinística; blocos de texto claro idênticos produzem blocos cifrados idênticos.
- Problemas adicionais:
  - Blocos idênticos ou mensagens com início igual são facilmente reconhecíveis.
  - A ordem dos blocos cifrados pode ser alterada sem que o receptor perceba.
- Conclusão: não recomendado para dados maiores que um bloco; alguns autores desaconselham completamente seu uso.

Modo ECB

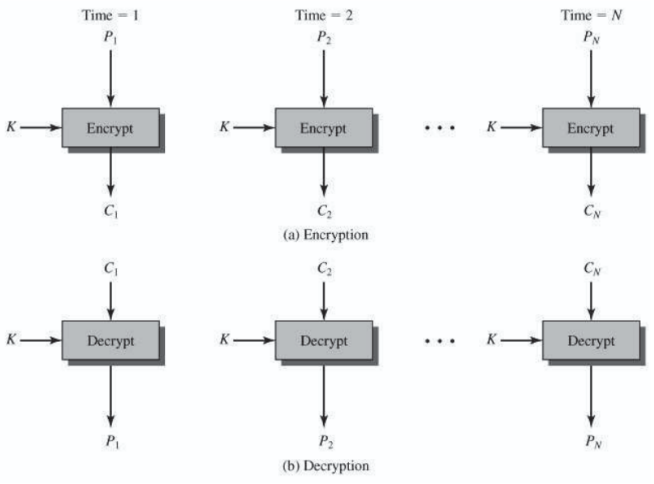


Figure 1: Scheme of the ECB mode of operation [2]

Cipher Block Chaining (CBC) – Considerações

- O modo CBC resolve o problema do ECB, reduzindo padrões repetidos no texto cifrado.
- Cada bloco de texto claro é XORado com o bloco cifrado anterior antes da encriptação.
- O primeiro bloco de texto claro é XORado com um *initialization vector* (IV).
- Benefício: mensagens longas com padrões repetidos podem ser tratadas de forma mais segura.
- Resultados de cifras distintas: mesmo texto claro cifrado múltiplas vezes gera diferentes textos cifrados devido ao IV.
- Desvantagem: requer mais tempo de processamento que o ECB devido ao encadeamento.
- Pode ser sincronizado para evitar propagação de erros causados por ruído no canal.
- O CBC não suporta paralelismo (**diferentemente do ECB)**

Modo CBC

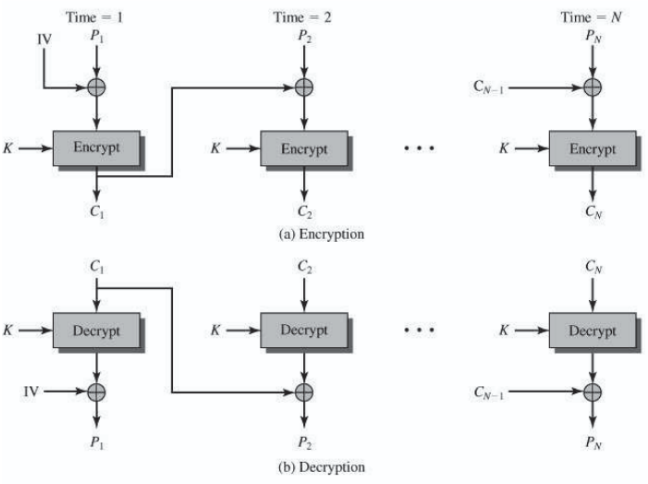
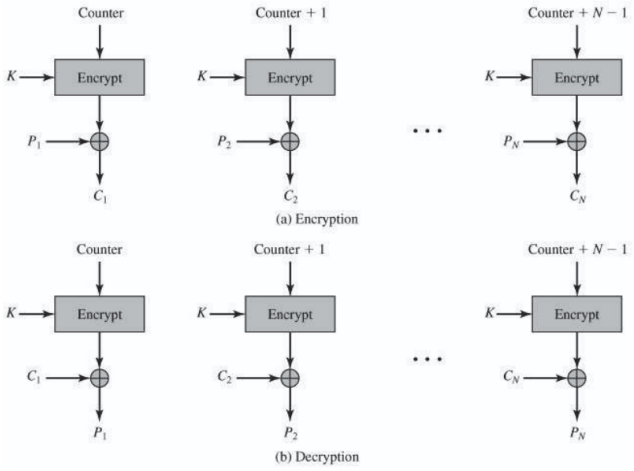


Figure 2: Scheme of the CBC mode of operation [2]

Counter Mode (CTR)

- Usa um contador como vetor de inicialização (IV), com tamanho igual ao do bloco.
- Cada bloco de texto claro é XORado com a saída da cifra aplicada ao contador.
- Não há necessidade de *padding* no último bloco.
- Os blocos são independentes, sem propagação de erro entre eles.
- Suporta paralelismo e pré-processamento, acelerando cifragem/decifragem.
- As operações de encriptação e decriptação são idênticas.
- É fundamental não reutilizar o mesmo contador com a mesma chave, sob risco de quebra completa da confidencialidade.
- Normalmente, o contador é inicializado com um valor único (ex.: 96 bits aleatórios + 32 bits incrementais).
- A chave deve ser trocada após $2^{n/2}$ blocos (onde n é o tamanho do bloco).
- Considerado um dos modos mais seguros e eficientes para AES.

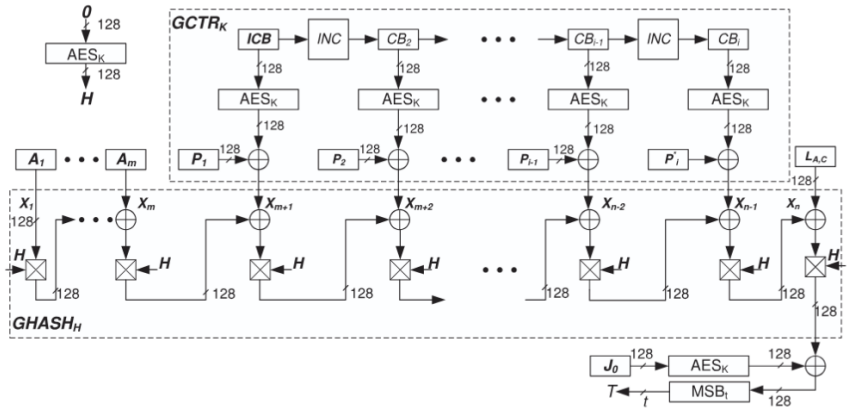
Modo CTR


Counter
Counter + 1
Counter + N − 1
K
Encrypt
P1
P2
PN
C1
C2
CN
(a) Encryption
Counter
Counter + 1
Counter + N − 1
K
Encrypt
C1
C2
CN
P1
P2
PN
(b) Decryption

AES – Galois/Counter Mode (GCM)

- GCM combina duas funções:
  - **Autenticação**: cálculo de um *tag* de integridade.
  - **Confidencialidade**: criptografia no modo *Counter*.
- O processo de confidencialidade é baseado no modo CTR (*Counter Mode*).
- A autenticação é realizada pela função **GHASH**, que usa multiplicações em $\mathrm{GF}(2^{128})$.
  - $GF(2^{128})$ é o campo de Galois com 128 bits: cada bloco de 128 bits é tratado como um polinômio de grau ≤ 127 com coeficientes 0 ou 1; a soma é XOR e a multiplicação é feita módulo
$$p(x) = x^{128} + x^7 + x^2 + x + 1.$$
- O *hash subkey* H é obtido aplicando AES sobre o bloco nulo.
- O *tag* de autenticação T é gerado a partir dos dados confidenciais e dos dados adicionais autenticados (AAD).
- Na decriptação autenticada, o *tag* é verificado para garantir integridade e autenticidade.
- Artigo: Modo GCM

Modo GCM


0
128
AES_K
128
H
GCTR_K
ICB
INC
CB_2
CB_{n-1}
INC
CB_n
128
AES_K
A_1
A_m
P_1
P_2
P_{n-1}
P*_n
L_{A,C}
X_1
X_m
X_{m+1}
X_{m+2}
X_{n-2}
X_{n-1}
X_n
H
GHASH_H
J_0
AES_K
MSB_t
T
t

AES-GCM: Campo de Galois e GHASH

- O GCM usa o campo finito $GF(2^{128})$ para autenticação.
- Cada bloco de 128 bits é tratado como um polinômio de grau até 127:

$$b_0 + b_1 x + b_2 x^2 + \cdots + b_{127} x^{127}, \quad b_i \in \{0, 1\}$$

- A chave de hash H é gerada cifrando um bloco de zeros com AES.
- O GHASH combina cada bloco de dados ou AAD com H assim:

$$X_i = (X_{i-1} \oplus B_i) \cdot H \mod p(x)$$

- O polinômio irreducível usado é fixo pelo padrão NIST:

$$p(x) = x^{128} + x^7 + x^2 + x + 1$$

Fluxo de autenticação no AES-GCM (GHASH)

O fluxo de autenticação no AES-GCM (GHASH) funciona assim:

1. Pega o acumulador do bloco anterior X (ou zero no primeiro bloco).
2. Faz XOR com o bloco atual da mensagem P.
3. Multiplica o resultado pelo H, que é a chave de hash obtida cifrando um bloco de zeros com AES.
4. Reduz módulo o polinômio

$$p(x) = x^{128} + x^7 + x^2 + x + 1$$

para manter 128 bits.

Ou seja, cada passo é:

$$X_i = ((X_{i-1} \oplus P_i) \cdot H) \bmod p(x)$$

- O XOR encadeia os blocos e mistura os dados.
- O módulo garante que o resultado continue com 128 bits, pronto para o próximo bloco ou para gerar a Tag final.

Aula 10 - Funções Hash

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação
Universidade Federal do Rio de Janeiro
gabrielaquino@ic.ufrj.br

Compilado em:
September 19, 2025

Funções de Hash

- Uma função de hash aceita uma mensagem de tamanho variável M como entrada.
- Produz um valor de tamanho fixo $h = H(M)$.
- O valor h é chamado de **hash** ou **digest**.

Propriedades Desejáveis de Hash

- A saída deve parecer **aleatória** e estar **uniformemente distribuída**.
- Uma pequena mudança em M altera, com alta probabilidade, muitos bits de h.
- Principal objetivo: **integridade de dados**.
- Exemplo: se qualquer bit de M for alterado, o hash $H(M)$ também mudará.

Função de Hash Criptográfica

- Tipo especial de função de hash usada em aplicações de segurança.
- Deve ser **computacionalmente inviável** de quebrar com eficiência maior que força bruta.
- Usada para verificar se os dados foram alterados.

Propriedades de uma Função de Hash Criptográfica

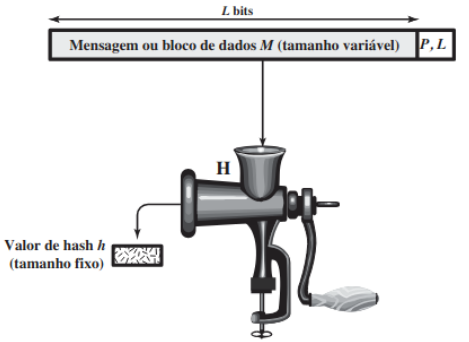
- **Mão única (one-way):** Dado um hash h, é inviável encontrar uma mensagem M tal que $H(M) = h$.
- **Livre de colisão (collision-free):** É inviável encontrar duas mensagens M_1 e M_2 tais que $H(M_1) = H(M_2)$.

Preenchimento em Funções de Hash

- Funções de hash processam mensagens em blocos de tamanho fixo.
- Quando a mensagem não é múltiplo do tamanho do bloco, adiciona-se **preenchimento** (padding).
  - Mensagem preenchida até se tornar um múltiplo de um tamanho fixo (ex: 1024 bits).
- O preenchimento inclui o **tamanho original da mensagem** em bits.
  - **Objetivo:** dificultar que um atacante crie uma mensagem alternativa com o mesmo hash.
- Garante que cada mensagem de tamanho diferente resulte em um hash único e seguro.

Preenchimento em Funções de Hash

Figura 11.1 Função de hash criptográfica; $h = \mathrm{H}(M)$.



P, L = preenchimento mais campo de tamanho

Aplicações de Funções de Hash Criptográficas

Uso do Hash:

- Ela é usada em diversas aplicações de segurança e protocolos da Internet.
- Talvez o hash seja o algoritmo criptográfico mais versátil.

Uso onde a Hash é empregada:

- Autenticação de mensagem
- Assinaturas digitais
- Arquivo de senha de mão única
- Detecção de intrusão e detecção de vírus
- Função pseudoaleatória (PRF)
- Gerador de número pseudoaleatório (PRNG)

Autenticação de Mensagem

- Autenticação de mensagem é um mecanismo usado para verificar a integridade de uma mensagem
- Garante que os dados recebidos estão exatamente como foram enviados, sem modificação, inserção, exclusão ou repetição.
- Em muitos casos, também é exigido que a identidade declarada do emissor seja validada.
- Em muitas aplicações, o valor gerado pelo Hash é chamado de **resumo de mensagem** (ou *digest*, em inglês).

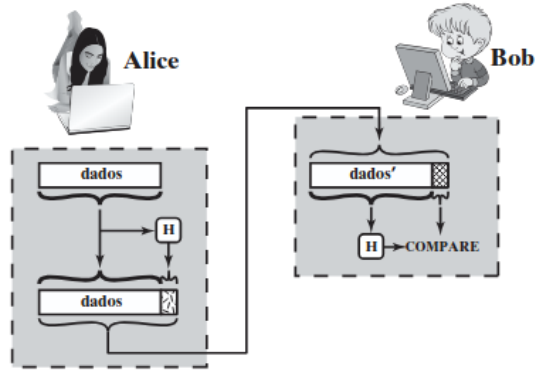
Autenticação de mensagem

A essência do uso de uma função de hash para autenticação de mensagem é a seguinte:

1. O emissor calcula um valor de hash como função dos bits da mensagem.
2. O emissor transmite a mensagem junto com o valor de hash.
3. O receptor recalcula o valor de hash sobre a mensagem recebida.
4. O receptor compara o valor calculado com o valor recebido.

Se houver divergência, o receptor sabe que a mensagem (ou o valor de hash) foi alterada.

Autenticação básica



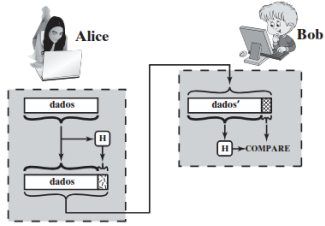
(a) Uso da função de hash para verificar integridade de dados

Pergunta

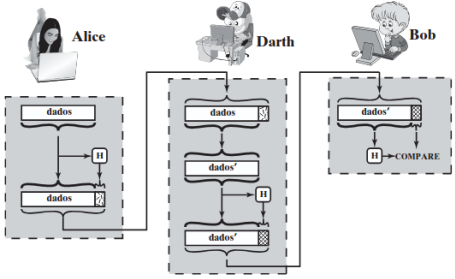
Qual o problema neste cenário?

Autenticação básica - Problema

Figura 11.2 Ataque contra função de hash.



(a) Uso da função de hash para verificar integridade de dados



(b) Ataque *man-in-the-middle*

Proteção do valor de hash

A função de hash precisa ser **transmitida de forma segura**.

- Se um adversário **alterar ou substituir** a mensagem, não deve ser viável alterar também o valor de hash para enganar o receptor.
- Exemplo de ataque:
  1. Alice transmite dados com o hash
  2. Darth intercepta, altera a mensagem e calcula um novo hash
  3. Bob recebe sem perceber a modificação.
- Para impedir esse ataque, **o valor de hash gerado por Alice precisa ser protegido**.

Pergunta

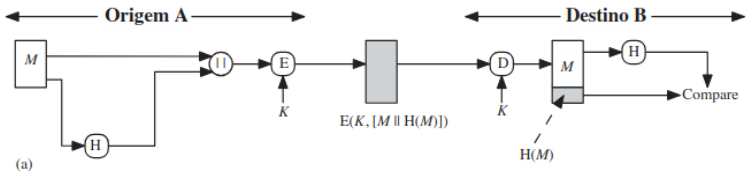
Como podemos proteger o Hash nesse cenário?

Métodos de proteção da Hash

- Método A: Autenticação com hash + cifragem simétrica
- Método B: Cifrando o hash com cifragem simétrica
- Método C: Mensagem com Hash e Valor Secreto
- Método D: Mensagem com Hash mais Valor Secreto com confidencialidade

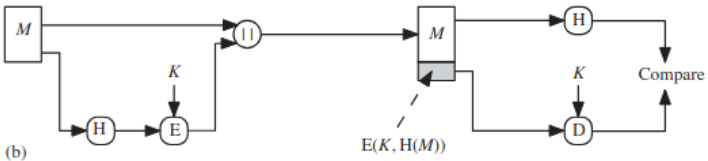
Método A: Autenticação com hash + cifragem simétrica

- Mensagem mais o código de hash concatenado **são encriptados** usando a encriptação simétrica.
- Como somente A e B compartilham a chave secreta, a mensagem deverá ter vindo de A e sem alteração.
- O código de hash oferece a estrutura ou redundância exigida para conseguir a autenticação.
- Como a encriptação é aplicada à mensagem inteira mais o código de hash, a confidencialidade também é fornecida.



Método B: Cifrando o hash com cifragem simétrica

- Somente o código de hash é encriptado, usando a encriptação simétrica.
- Isso reduz o peso do processamento para as **aplicações que não exigem confidencialidade**.



Pergunta

Qual o motivo de passar a mensagem em texto plano?

Vantagens do Uso de Hash mas sem a cifragem da mensagem

- Quando a **confidencialidade não é exigida**:
  - usar apenas hash (método com valor secreto) requer menos cálculos que cifrar a mensagem inteira.
  - O software de encriptação é relativamente lento, especialmente com fluxo constante de mensagens.
  - Custos de hardware de encriptação podem ser altos; chips de baixo custo existem, mas cada nó precisa ter capacidade.

Exemplo desse cenário

Baixando a .iso do Linux Mint e verificando com GPG

O que o GPG faz com esses dois arquivos?

Comando:

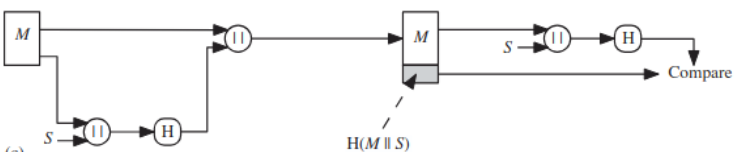
```
gpg --verify sha256sum.txt.gpg sha256sum.txt
```

Etapas realizadas pelo GPG:

1. Lê a assinatura digital contida em `sha256sum.txt.gpg`.
2. Calcula o hash real do conteúdo atual de `sha256sum.txt`.
3. Compara o hash calculado com o hash assinado.
   - Se forem **iguais**: a assinatura é válida (`Good signature`).
   - Se forem **diferentes**: a assinatura falha (`BAD signature`).

Método C: Mensagem com Hash e Valor Secreto

- Usar apenas uma função de hash, sem cifragem, para autenticação de mensagem.
- Ambas as partes compartilham um valor secreto comum: S.
- Funcionamento:
  1. Emissor (A) calcula o hash sobre a concatenação da mensagem e do segredo: $H(M \,\|\, S)$.
  2. O valor de hash resultante é anexado à mensagem e enviado a B.
  3. Receptor (B), possuindo S, recalcula o hash para verificar a integridade e autenticidade.
- Como S não é enviado, um adversário não consegue modificar a mensagem nem gerar mensagens falsas.



Código de Autenticação de Mensagem (MAC)

Método C é a base do HMAC - Hash Based Message Authentication Code

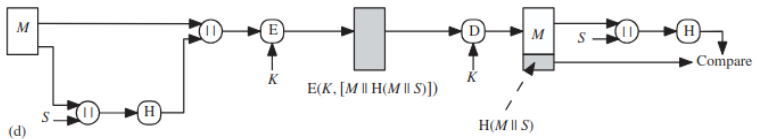
- A autenticação de mensagem normalmente é alcançada usando um **MAC** (Message Authentication Code), também chamado de função de hash chaveada.
- MACs são usados entre duas partes que compartilham uma **chave secreta** para autenticar informações trocadas.
- A função MAC recebe como entrada a chave secreta e um bloco de dados, produzindo um valor de hash (**MAC**) associado à mensagem.

Código de Autenticação de Mensagem (MAC)

- Para verificar integridade, aplica-se novamente a função MAC sobre a mensagem e compara-se com o MAC recebido.
- Um invasor que altere a mensagem não poderá gerar o MAC correto sem conhecer a chave secreta.
- A verificação também garante autenticidade: apenas a parte que conhece a chave secreta pode ter gerado o MAC.

Método D: Mensagem com Hash mais Valor Secreto com confidencialidade

- A **confidencialidade** pode ser acrescentada à abordagem do método anterior encriptando a mensagem inteira mais o código de hash

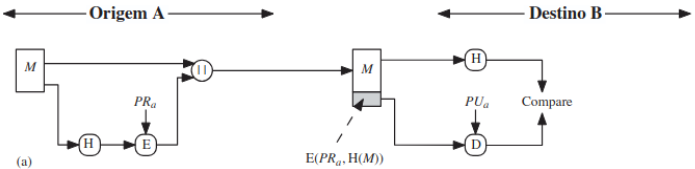


Cenário

Esse cenário mostra exatamente um canal criptografado com a autenticacao da mensagem, igual na VPN!

Assinaturas Digitais

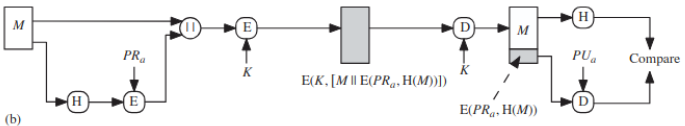
- **Usa Chave pública e privada**
- Assinatura digital é uma aplicação importante, similar à autenticação de mensagem.
- O valor de hash da mensagem é encriptado com a **chave privada** do usuário.
- Qualquer pessoa que conheça a **chave pública** do usuário pode verificar a integridade da mensagem associada à assinatura.
- Um invasor que tente alterar a mensagem precisaria conhecer a chave privada do usuário.



Esse caso é exatamente o caso do hash da iso do Linux Mint!

Assinaturas Digitais com Confidencialidade

- Se, além da assinatura digital, o que se procura é confidencialidade, então a mensagem mais o código hash encriptado com a chave privada pode ser encriptado usando uma chave secreta simétrica. Essa é uma técnica comum.



Arquivos de Senha de Mão Única

- Funções de hash são usadas para criar arquivos de senha de **mão única**.
  - No linux usa-se o /etc/shadow
- Em vez de armazenar a senha real, o sistema operacional armazena o **hash da senha**.
- Assim, mesmo que um hacker acesse o arquivo, a senha real não pode ser recuperada.
- Processo de autenticação:
  1. O usuário fornece a senha
  2. O sistema compara o **hash informado** com o hash armazenado.
- Este método é amplamente usado na maioria dos sistemas operacionais.

Detecção de Intrusão e Vírus com Hash

- Funções de hash podem ser usadas para **detecção de intrusão** e **detecção de vírus**.
- Armazene $H(F)$ para cada arquivo em um sistema e guarde os valores de hash de forma segura.
- Posteriormente, verifique se um arquivo foi modificado recalculando $H(F)$.
- Um intruso precisaria alterar F sem alterar $H(F)$ para evitar detecção, o que é computacionalmente inviável.

Trabalho interessante:

- Documentação Labrador
- Código Labrador

Além disso, Hashes são usadas em função pseudoaleatória (PRF) ou gerador de número pseudoaleatório (PRNG)

Funções de Hash Simples

- Para entender considerações de segurança, apresentamos funções de hash simples e **não seguras**.
- Todas as funções de hash operam sobre blocos de n bits da entrada (mensagem, arquivo etc.).
- A entrada é processada bloco a bloco em um padrão iterativo para produzir um hash de n bits.
- Um exemplo simples: o **XOR bit a bit** de cada bloco.

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

onde

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

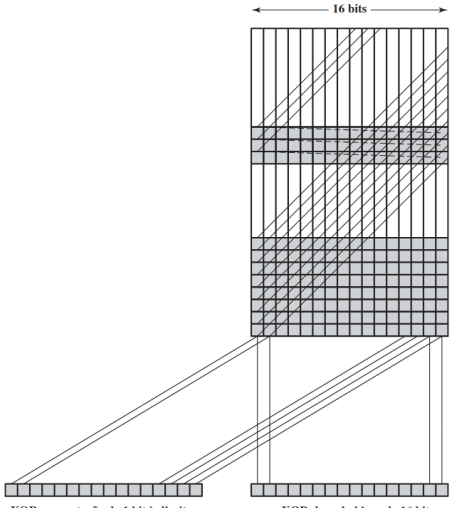
$m =$ número de blocos de n bits na entrada

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

$\oplus =$ operação XOR

Esquema de Hash simples com XOR

Figura 11.5 Duas funções de hash simples.



Limitações do XOR em Funções de Hash

- XOR simples não é suficiente quando apenas o código de hash é encriptado.
- Problema: blocos de texto cifrado podem ser **reordenados** sem alterar o valor do hash.
- Isso permite que um atacante modifique a mensagem sem que a integridade seja detectada.
- Conclusão: para garantir a integridade, precisamos de funções de hash criptográficas **não lineares e resistentes a colisões**.

Pré-imagem e Colisões em Funções de Hash

- Para um valor de hash $h = H(x)$, x é chamado de **pré-imagem** de h.
- Isso significa que x é um bloco de dados cuja função hash produz h.
- Funções hash são **mapas muitos-para-um**: para qualquer valor h, podem existir várias pré-imagens.
- Uma **colisão** ocorre se existirem $x \neq y$ tal que $H(x) = H(y)$.
- Colisões são indesejáveis quando funções de hash são usadas para **integridade de dados**.

Pré-imagens e Potenciais Colisões

- Suponha uma função de hash H com saída de n bits e entrada de b bits ($b > n$).
- Total de mensagens possíveis: 2^b.
- Total de valores de hash possíveis: 2^n.
- Em média, cada valor de hash corresponde a 2^{b-n} pré-imagens.
- Se H distribui uniformemente os valores de hash, cada hash terá aproximadamente 2^{b-n} pré-imagens.
- Para entradas de tamanho variável, a variação de pré-imagens por valor de hash aumenta.
- Apesar disso, os riscos de segurança não são tão graves; é necessário definir requisitos precisos de segurança para funções de hash criptográficas.

Requisitos de Funções de Hash

Tabela 11.1 Requisitos para função de hash criptográfica H.

Requisito	Descrição
Tamanho de entrada variável	H pode ser aplicado em um bloco de dados de qualquer tamanho.
Tamanho da saída fixo	H produz uma saída de tamanho fixo.
Eficiência	$H(x)$ é relativamente fácil de calcular para qualquer valor de x informado, através de implementações tanto em hardware quanto em software.
Resistência à pré-imagem (propriedade de mão única)	Para qualquer valor de hash h informado, é computacionalmente impossível encontrar y, de modo que $H(y) = h$.
Resistência à segunda pré-imagem (resistência à colisão fraca)	Para qualquer bloco x informado, é computacionalmente impossível encontrar $y \neq x$ com $H(y) = H(x)$.
Resistência à colisão forte	É computacionalmente impossível encontrar qualquer par (x, y), de modo que $H(x) = H(y)$.
Pseudoaleatoriedade	A saída de H atende os testes padrão de pseudoaleatoriedade.

As primeiras três propriedades são requisitos para a aplicação prática de uma função de hash.

Resistência à Pré-imagem (Propriedade de Mão Única)

- A resistência à pré-imagem significa que é fácil gerar o código de hash a partir da mensagem.
- Porém, é praticamente impossível gerar a mensagem a partir do código de hash.
- Essencial quando a autenticação envolve um valor secreto que não é transmitido.
- Se a função de hash não tiver esta propriedade, um invasor pode:
  - Observar a mensagem M e o hash $h = H(S||M)$.
  - Inverter a função de hash para obter $S||M = H^{-1}(h)$.
  - Recuperar o valor secreto S facilmente.
- Portanto, a resistência à pré-imagem é crucial para proteger valores secretos.

Resistência à Segunda Pré-imagem

- Garante que é impossível encontrar uma mensagem alternativa com o mesmo valor de hash de uma mensagem específica.
- Importante para prevenção contra falsificação quando se usa um hash encriptado.
- Se a função de hash não possuir essa propriedade, um invasor poderia:
  - Observar ou interceptar uma mensagem com seu hash encriptado.
  - Decriptar o hash da mensagem original.
  - Criar uma nova mensagem diferente com o mesmo hash.
- Portanto, esta propriedade protege contra ataques de falsificação.

Funções Hash Fraca e Forte

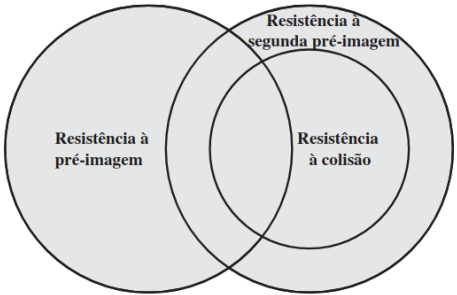
- Uma função de hash que satisfaz as primeiras cinco propriedades é chamada de **função hash fraca**.
- Se a função também satisfaz a sexta propriedade, **resistência à colisão**, é chamada de **função hash forte**.
- Função hash forte protege contra ataques onde terceiros tentam gerar mensagens diferentes com o mesmo hash.
- Exemplo de ataque sem resistência à colisão:
  - Bob cria duas mensagens diferentes com o mesmo hash (m1 e m2).
  - Alice assina a mensagem m1.
  - Bob usa o hash da mensagem m1 para reivindicar que a mensagem m2 foi assinada.
- Portanto, a resistência à colisão é crucial para evitar falsificação de assinaturas.

Pseudoaleatoriedade em Funções de Hash Criptográficas

- A pseudoaleatoriedade não é tradicionalmente citada como requisito formal, mas é implicitamente importante.
- Funções de hash criptográficas são frequentemente usadas para:
  - Derivação de chaves.
  - Geração de números pseudoaleatórios (PRNG/PRF).
- Nas aplicações de integridade de mensagens, as propriedades de resistência dependem de a saída parecer aleatória.
- Portanto, é apropriado considerar que uma função de hash produz uma saída pseudoaleatória.

Relação entre propriedades de Funções de Hash

Figura 11.6 Relação entre as propriedades das funções de hash.



- Uma função que é resistente à colisão também é resistente à segunda pré-imagem, mas o inverso não é necessariamente verdadeiro.
- Uma função pode ser resistente à colisão, mas não ser resistente à pré-imagem, e vice-versa. Uma função pode ser resistente à pré-imagem, mas não ser resistente à segunda pré-imagem e vice-versa.

Propriedades de resistência Funções de Hash e seu uso

Tabela 11.2 Propriedades de resistência necessárias para várias aplicações de integridade de dados.

	Resistência à pré-imagem	Resistência à segunda pré-imagem	Resistência à colisão
Hash + assinatura digital	sim	sim	sim*
Detecção de intrusão e detecção de vírus		sim	
Hash + encriptação simétrica			
Arquivo de senha de mão única	sim		
MAC	sim	sim	sim*

*Resistência necessária se o invasor é capaz de elaborar um determinado ataque de mensagem

Ataques de Força Bruta em Funções de Hash

- Ataques de força bruta não dependem do algoritmo de hash, apenas do tamanho do valor de hash em bits.
- Consiste em tentar todas as combinações possíveis até encontrar uma pré-imagem ou colisão.
- O esforço necessário cresce exponencialmente com o tamanho do hash: para um hash de n bits, há 2^n possíveis valores.
- Diferente da criptoanálise, que explora vulnerabilidades específicas do algoritmo.
- Exemplo: para um hash de 128 bits, seriam necessárias 2^{128} tentativas em média para encontrar uma colisão por força bruta.

Ataques de Pré-imagem e Segunda Pré-imagem

Cenário: O atacante conhece uma saída da Hash

- O adversário busca um valor y tal que $H(y) = h$, onde h é um hash conhecido.
- Método de força bruta: testar valores aleatórios de y até encontrar uma correspondência.
- Para um hash de m bits, o esforço médio necessário é 2^{m-1} tentativas.
- Esse ataque explora a dificuldade de inverter a função de hash (resistência à pré-imagem).

Ataques Resistentes à Colisão

Cenário: O atacante busca desobrir pares de saída da Hash

- O adversário busca duas mensagens x e y tal que $H(x) = H(y)$.
- Esse ataque exige **menos esforço** do que um ataque de pré-imagem ou segunda pré-imagem.
- Baseado no **paradoxo do aniversário**: a probabilidade de colisão aumenta rapidamente com o número de tentativas.
- Para um hash de m bits, o esforço médio para encontrar uma colisão é aproximadamente $2^{m/2}$ tentativas.

Ataque de Colisão Explorado via Paradoxo do Aniversário

Estratégia para explorar o paradoxo do dia do aniversário:

- Preparação da origem A: mensagem legítima x é criada
- Oponente gera $2^{m/2}$ variações x' de x com mesmo significado e armazena os hashes.
- Oponente prepara uma mensagem fraudulenta y para a qual deseja a assinatura.
- Pequenas variações y' de y são geradas; o oponente calcula $H(y')$ e verifica correspondência com algum $H(x')$.
- Quando há correspondência, a variação válida de A é usada para assinatura, que é então aplicada à variação fraudulenta y'.

Ambas produzem a mesma assinatura.

Exemplo de Ataque de Colisão com Hash de 64 bits

- Com um hash de 64 bits, o esforço necessário é da ordem de 2^{32}.
- Criação de variações que mantêm o mesmo significado não é difícil.
- Exemplos de variações:
  - Inserção de pares de caracteres “espaço-espaço-retrocesso” entre palavras.
  - Substituição de “espaço-retrocesso-espaço” em posições selecionadas.
  - Reescrita da mensagem mantendo o significado original.

Resumo do esforço exigido

Resumo: Para um código de hash de tamanho m, o nível de esforço exigido, conforme vimos, é proporcional ao seguinte:

Resistência à pré-imagem	2^m
Resistência à segunda pré-imagem	2^m
Resistência à colisão	$2^{m/2}$

Resistência a Colisões em Hashes

- A resistência à colisão é desejável em códigos de hash seguros.
- Para um hash de m bits, a força contra ataques por força bruta é aproximadamente $2^{m/2}$.
- Van Oorschot e Wiener [VANO94] projetaram uma máquina de US$10 milhões para MD5 (128 bits):
  - Capaz de encontrar uma colisão em 24 dias.
  - Mostra que 128 bits é inadequado para segurança moderna.
- Hashes de 160 bits (como SHA-1) aumentam a resistência:
  - Mesma máquina levaria mais de 4.000 anos para encontrar uma colisão.
  - Contudo, com tecnologia atual, 160 bits começa a ficar suspeito.

Criptoanálise de Funções de Hash e MACs

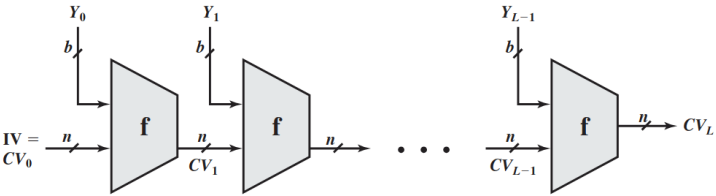
- Assim como em algoritmos de criptografia, ataques criptoanalíticos em **hashes** e **MACs** buscam explorar propriedades do algoritmo para superar a simples busca exaustiva.
- A resistência de um hash ou MAC à criptoanálise é medida comparando-se seu esforço com o esforço de um ataque por força bruta.
- Um hash ou MAC ideal exigirá **esforço criptoanalítico maior ou igual ao esforço por força bruta**.

Estrutura Iterativa de Funções de Hash

- A estrutura iterativa de hash foi proposta por Merkle [MERK79, MERK89] e é usada na maioria das funções de hash atuais, incluindo SHA.
- Funcionamento geral:
  - A mensagem de entrada é dividida em L blocos de tamanho fixo b bits.
  - Se necessário, o bloco final é preenchido para completar b bits e inclui o tamanho total da mensagem.
  - Incluir o tamanho dificulta ataques, pois o oponente deve encontrar colisões com mensagens do mesmo ou de tamanhos diferentes que levem ao mesmo hash.
- O algoritmo usa repetidamente uma **função de compactação** f:
  - Recebe duas entradas: a **variável de encadeamento** (n bits da etapa anterior) e o bloco atual (b bits). Normalmente, b $>$ n;
  - Produz uma saída de n bits.
  - A variável de encadeamento inicial é definida pelo algoritmo e o valor final dela é o **hash resultante**.

Estrutura geral de hash seguro

Figura 11.8 Estrutura geral do código de hash seguro.



IV = valor inicial
CV_i = variável de encadeamento
Y_i = i-ésimo bloco de entrada
f = algoritmo de compactação
L = Número de blocos de entrada
n = Tamanho do código de hash
b = Tamanho do bloco de entrada

Estrutura geral de hash seguro

A função de hash pode ser resumida da seguinte forma:

$$CV_0 = IV = \text{valor inicial de } n \text{ bits}$$
$$CV_i = \mathrm{f}(CV_{i-1}, Y_{i-1}) 1 \leq i \leq L$$
$$\mathrm{H}(M) = CV_L$$

onde a entrada da função de hash é uma mensagem M consistindo nos blocos Y0, Y1, ..., YL-1

Motivação e Criptoanálise de Funções de Hash

- Motivação da estrutura iterativa (Merkle [MERK89], Damgård [DAMG89]):
  - Se a função de compactação f for à prova de colisão, a função de hash iterativa resultante também será.
  - Permite criar hashes seguros para mensagens de qualquer tamanho.
  - O design seguro de uma função de hash se reduz ao design de uma função de compactação segura para blocos de tamanho fixo.
- Criptoanálise de funções de hash:
  - Foca na estrutura interna de f.
  - Ataques procuram produzir colisões eficientes para uma única execução de f, considerando o valor fixo do IV.
  - Normalmente, f consiste em várias rodadas, e o ataque analisa padrões de mudança de bits entre rodadas.

Colisões em Funções de Hash

- **Importante**: Para qualquer função de hash, **colisões sempre existem**:
  - Mensagens têm tamanho $\geq 2b$ (devido ao campo de tamanho)
  - Hashes têm tamanho fixo n, com $b > n$
- O objetivo de uma função de hash segura não é eliminar colisões (isso é impossível), mas torná-las **computacionalmente inviáveis de encontrar**.
- Assim, a segurança é definida pelo **esforço necessário para descobrir uma colisão**, não pela sua inexistência.

Secure Hash Algorithm (SHA)

- O **SHA** é a função de hash mais utilizada nos últimos anos.
- Em 2005, era praticamente o último algoritmo de hash padronizado restante após vulnerabilidades em outros algoritmos.
- Desenvolvido pelo **NIST** e publicado como padrão federal (**FIPS 180**) em 1993.
- Primeira versão (**SHA-0**) apresentou vulnerabilidades criptoanalíticas.
- Revisão lançada em 1995 (**FIPS 180-1**), conhecida como **SHA-1**.
- Baseado na função de hash **MD4**, seguindo de perto seu projeto.

SHA-1 e SHA-2

- **SHA-1** produz um hash de 160 bits.
- Em 2002, o **NIST** revisou o padrão (**FIPS 180-2**), definindo três novas versões:
  - SHA-256 (256 bits)
  - SHA-384 (384 bits)
  - SHA-512 (512 bits)
- Coletivamente, estas versões são conhecidas como **SHA-2**.
- Mantêm a mesma estrutura básica do SHA-1, usando aritmética modular e operações binárias lógicas.
- Em 2008, o **FIPS PUB 180-3** adicionou uma versão de 224 bits (SHA-224).
- SHA-1 e SHA-2 também são especificados no **RFC 6234**, que inclui implementação em C.

Descontinuação do SHA-1

- Em 2005, o **NIST** anunciou a intenção de retirar a aprovação do SHA-1 e adotar o SHA-2 por volta de 2010.
- Uma equipe de pesquisa demonstrou um ataque que poderia gerar duas mensagens diferentes com o mesmo hash SHA-1 usando 2^{69} operações.
- Este número é significativamente menor que as 2^{80} operações anteriormente estimadas para encontrar uma colisão.
- O resultado acelerou a necessidade de transição para SHA-2.
- Referência: Wang et al. [WANG05].

Tabela SHA

Tabela 11.3 Comparação de parâmetros do SHA.

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Tamanho do resumo da mensagem	160	224	256	384	512
Tamanho da mensagem	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Tamanho do bloco	512	512	512	1024	1024
Tamanho da word	32	32	32	64	64
Número de etapas	80	64	64	80	80

Nota: todos os tamanhos são medidos em bits.

SHA-512: Etapa 1 - Preenchimento

- Entrada: mensagem com tamanho menor que 2^{128} bits.
- Saída: resumo (hash) de 512 bits.
- Processamento em blocos de 1024 bits.
- **Etapa 1 - Preenchimento:**
  - A mensagem é preenchida para que o tamanho seja congruente a 896 módulo 1024.
  - O preenchimento é sempre aplicado, mesmo se a mensagem já tiver o tamanho desejado.
  - Número de bits de preenchimento: entre 1 e 1024.
  - Estrutura do preenchimento: um bit 1 seguido pelos bits 0 necessários.

SHA-512: Etapa de Anexar Tamanho

- **Etapa 2 - Anexar tamanho:**
  - Um bloco de 128 bits é anexado à mensagem.
  - O bloco contém o tamanho da mensagem original (antes do preenchimento) como um inteiro de 128 bits sem sinal.
  - Ordem dos bytes: byte mais significativo primeiro.
- Após as duas primeiras etapas, a mensagem resultante tem comprimento múltiplo de 1024 bits.
- Mensagem expandida representada como blocos de 1024 bits: $M_1, M_2, \ldots, M_N$.
- Tamanho total da mensagem expandida: $N \times 1024$ bits.

SHA-512: Inicialização do Buffer de Hash

- Um buffer de 512 bits mantém resultados intermediários e finais.
- Representado por 8 registradores de 64 bits: a, b, c, d, e, f, g, h.
- Inicialização com valores hexadecimais:

```
a = 6A09E667F3BCC908   e = 510E527FADE682D1
b = BB67AE8584CAA73B   f = 9B05688C2B3E6C1F
c = 3C6EF372FE94F82B   g = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1   h = 5BE0CD19137E2179
```

- Valores armazenados em **big-endian**.
- Derivados dos primeiros 64 bits das partes fracionárias das raízes quadradas dos oito primeiros números primos.