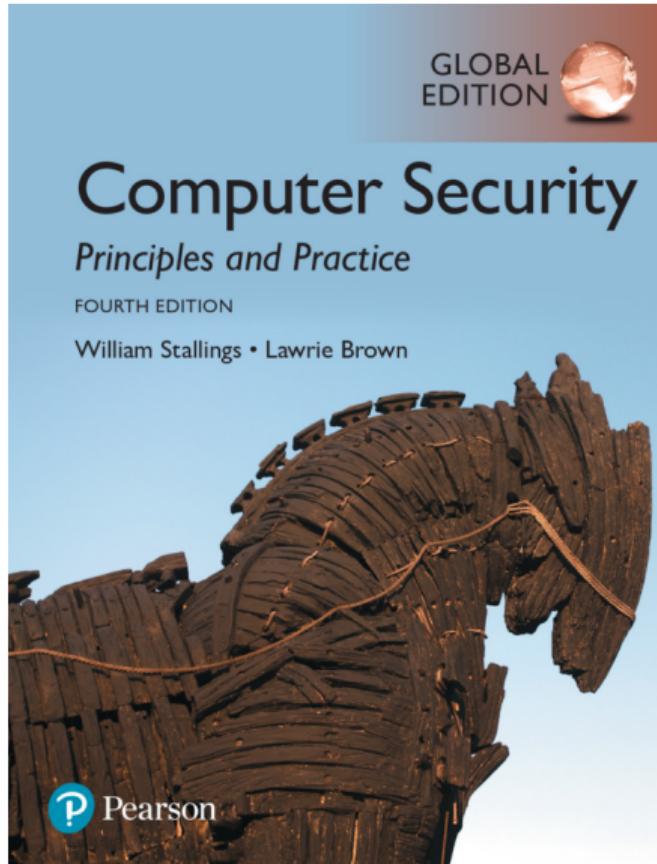


# ICP473 - Segurança da Informação

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação  
Universidade Federal do Rio de Janeiro  
[gabrielaquino@ic.ufrj.br](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 2025



- Computer Security - Principles and Practice  
Fourth Edition  
Global Edition  
William Stallings  
Lawrie Brown  
ISBN 10: 1-292-22061-9  
ISBN 13: 978-1-292-22061-1

William Stallings

- Criptografia e segurança de redes:  
princípios e práticas  
William Stallings;  
tradução Daniel Vieira; revisão técnica  
Paulo Sérgio Licciardi Messeder  
Barreto, Rafael Misoczki. – 6. ed. –  
São Paulo: Pearson  
Education do Brasil, 2015.  
Título original: Cryptography and  
network security  
ISBN 978-85-430-1450-0



# Método de Avaliação

## Componentes da Nota

- **Provas (P1 e P2):** Média das provas (MP) → 80% da nota
- **Trabalhos:** Média dos trabalhos (MT) → 20% da nota

## Critério de Aprovação Direta

- Média ponderada de MP e MT  $\geq 7$  → Aprovado direto
- Se  $3 \leq$  Média ponderada de MP e MT  $<= 7$  → Prova final (PF)

## Cálculo da Média Final

$$\text{Média final} = \frac{\text{Média anterior} + \text{PF}}{2}$$

- Se Média final  $\geq 5 \Rightarrow$  Aprovado
- Caso contrário  $\Rightarrow$  Reprovado

# Aula 1 - Motivação Inicial

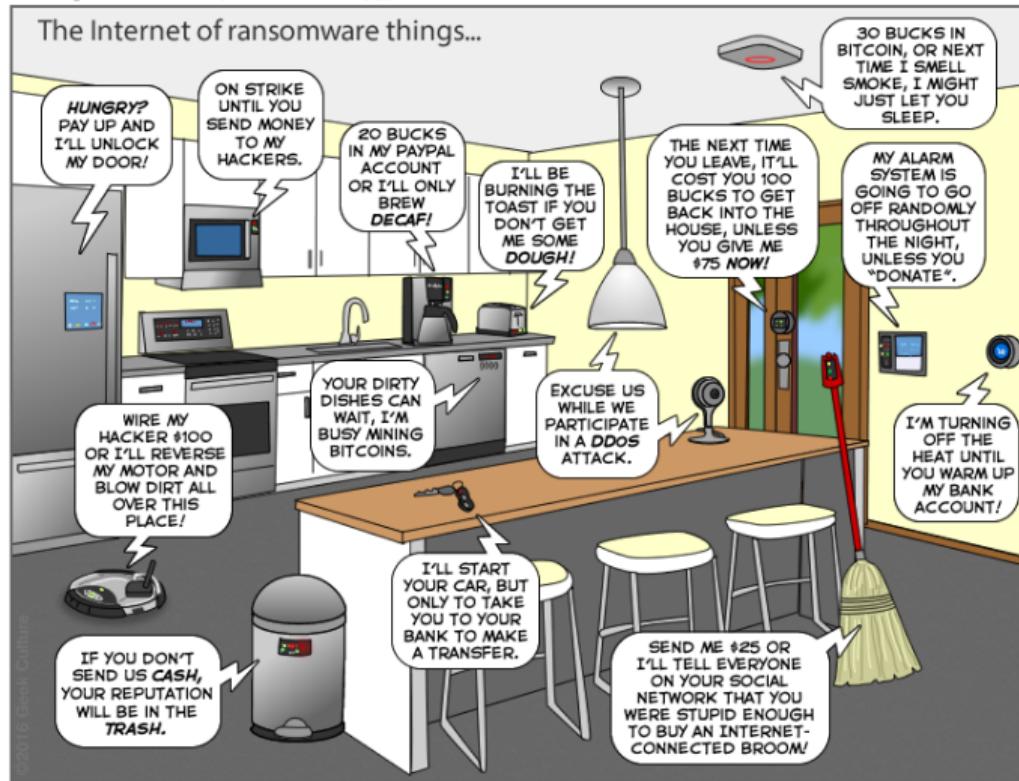
Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação  
Universidade Federal do Rio de Janeiro  
[gabrielaquino@ic.ufrj.br](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 2025

# Preocupações da atualidade

The Joy of Tech™ by Nitrozac & Snaggy



You can help us keep the comics coming by becoming a patron!  
[www.patreon/joyoftech](http://www.patreon/joyoftech)

[joyoftech.com](http://joyoftech.com)

# Informação

## Ativo valioso

A informação é o ativo mais valioso para uma organização ou para uma pessoa.

## Por que?

- **tomada de decisão.**
- **vantagem competitiva.**
- **ativo financeiro, estratégico ou pessoal.**

## Impactos da perda ou comprometimento

- Prejuízos financeiros.
- Danos à reputação.
- Risco jurídico.

# Preocupações da atualidade

Matheus @matheus\_gneto · 37 min  
Até o servidor do Gmail está com preguiça hoje.

10:04 4G

Google

502. That's an error.  
The server encountered a temporary error and could not complete your request.  
Please try again in 30 seconds. That's all we know.

946

thay @aquaryane · 48 min  
poxa o gmail saiu do ar infelizmente não posso mais trabalhar hoje

A Vida É Um Moinho @SrMillionerd · 48 min  
Gmail foi de arrasta pra cima? 😱

Foul tarnished @mariocaraujo · 48 min  
ah não gmail, não sacaneia em plena segunda de manhã

Pedro Nabuco @Nabuco · 48 min  
Gmail mais sequelado do que eu que fui a bloco ontem

Daniel Fernandes @danfernandesg · 48 min  
Gmail deu ruim.

Gabriel Gobg @\_GabrielGobg · 48 min  
em plena segunda e o Gmail foi de base

Link: [Notícia](#)

# Preocupações da atualidade

yahoo/news

## Microsoft outage: CrowdStrike security update impacts airports, hospitals, banks around the world

A CrowdStrike security update was automatically installed across Microsoft servers early Friday morning.

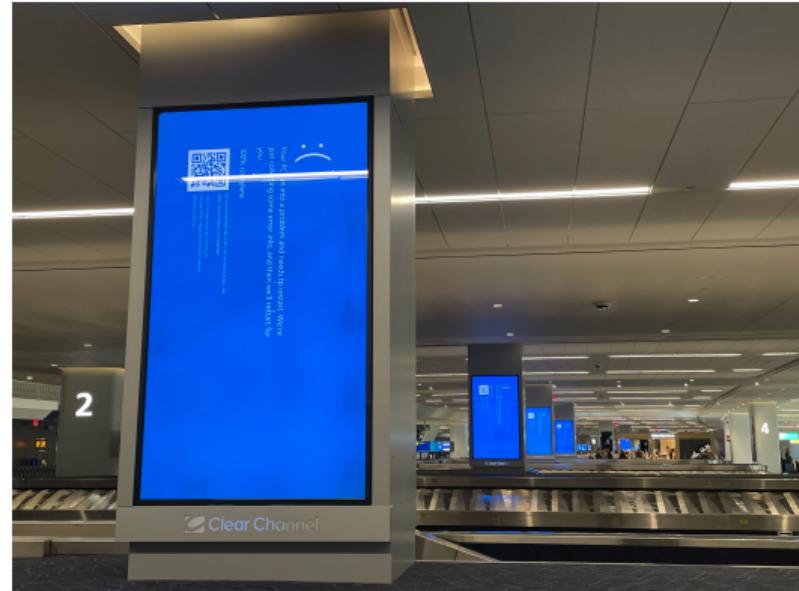
 Katie Mather, Reporter

Updated Fri, July 19, 2024 at 2:33 PM GMT-3

6 min read



02.3k



Link: [Notícia](#)

# Preocupações da atualidade

## Megavazamento de dados de 223 milhões de brasileiros: o que se sabe e o que falta saber

Número é maior do que a população do país, estimada em 212 milhões, porque inclui dados de falecidos. Informações expostas incluem CPF, nome, sexo e data de nascimento, além de uma tabela com dados de veículos e uma lista com CNPJs. Origem dos dados ainda é desconhecida.

Por G1

28/01/2021 18h34 · Atualizado há 4 anos

### Quais dados vazaram?

Foram dois vazamentos:

Um deles tinha **223 milhões de números de CPF**, acompanhado de informações como nome, sexo e data de nascimento, além de uma tabela com dados de veículos e uma lista com CNPJs (Cadastro Nacional de Pessoas Jurídicas). Essas informações circulam na Internet de forma gratuita.

O outro incluía, além dos 223 milhões de CPFs, informações sobre escolaridade, benefícios do INSS e programas sociais (como o Bolsa Família), renda, entre outras informações. Esse está sendo vendido por criminosos.

Juntos, os dois vazamentos continham:

- Dados básicos relativos ao CPF (nome, data de nascimento e endereço).
- Endereços.
- Fotos de rosto.
- Score de crédito (que diz se é bom pagador), renda, cheques sem fundo e outras informações financeiras.
- Imposto de renda de pessoa física.
- Dados cadastrais de serviços de telefonia.
- Escolaridade.
- Benefícios do INSS.
- Dados relativos a servidores públicos.
- Informações do LinkedIn.

Link: Notícia

# Caso real brasileiro

## Caso Real no Brasil — 2025

- Em julho de 2025, um ataque hacker contra a empresa C&M Software resultou no desvio de cerca de **R\$541 milhões** de uma instituição financeira.
- C&M Software provia a infraestrutura técnica para que seus clientes (bancos pequenos, fintechs, cooperativas de crédito, etc.) conseguissem se conectar ao Sistema de Pagamentos Instantâneos (SPI) e ao Sistema de Pagamentos Brasileiro (SPB) do Banco Central.

## Repercussão na imprensa:

[Link 1](#) ; [Link 2](#) ; [Link 3](#) ; [link 4](#)

## O que esse caso demonstra

- Ataque que usou o fator humano ( "insiders confiáveis" ) - Fator humano é crítico.
- Uma única falha de segurança pode resultar em **prejuízos bilionários**.

# Quanto Vale a Informação?

## Ideia central

O esforço despendido para proteger a informação depende diretamente do **valor dessa informação**.

## Reflexão

Se a informação for **crítica** para o negócio ou para a vida das pessoas, a proteção deve ser proporcionalmente **robusta**.

# Quanto Vale a Informação?

## Princípio Econômico da Segurança da Informação

O custo para se proteger contra uma ameaça deve ser **menor** que o custo de recuperação caso a ameaça se concretize. RFC 2196

Link: [RFC 2196 - Site Security Handbook](#)

## Reflexão

Investir em segurança não é gasto — é evitar perdas maiores.

Só que... no mundo real...



When you think you've closed all the security gaps,  
but your employee clicks on a phishing email!



# Aula 2 - Conceitos Básicos de Segurança da Informação

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação  
Universidade Federal do Rio de Janeiro  
[gabrielaquino@ic.ufrj.br](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 2025

## **Computer Security (COMPUSEC) (NISTIR 7298 - Link):**

Medidas e controles que garantem a **confidencialidade, integridade e disponibilidade** dos ativos de sistemas de informação, incluindo hardware, software, firmware e as informações processadas, armazenadas e comunicadas.

- **Confidencialidade dos dados:** Garante que informações privadas ou confidenciais não sejam disponibilizadas ou divulgadas a indivíduos não autorizados.
- **Privacidade:** Garante que os indivíduos tenham controle ou influência sobre quais informações relacionadas a eles podem ser coletadas, armazenadas e divulgadas, e por quem.

**Confidencialidade** é o sigilo de informações ou recursos.

A necessidade de manter informações em segredo surge do uso de computadores em áreas sensíveis - Ex: Governo, indústria e outras áreas.

- Instituições militares e civis do governo aplicam o princípio do "**necessário saber**".
- Empresas protegem projetos proprietários contra concorrência desleal.
- Todas as organizações mantêm registros pessoais em sigilo.

# Confidencialidade e Controle de Acesso

Mecanismos de **controle de acesso** são essenciais para garantir a confidencialidade.

Um dos principais mecanismos é a **criptografia**, que embaralha os dados para torná-los incompreensíveis sem a chave correta.

## Exemplo:

- Um sistema cifra uma declaração de imposto de renda.
- Apenas quem possui a **chave criptográfica** pode decifrá-la e acessar os dados.
- Porém, se a chave for exposta durante seu uso, a confidencialidade é comprometida.

*A proteção da chave é tão crítica quanto a proteção da própria informação.*

# Confidencialidade Além dos Dados

**Confidencialidade** não se aplica apenas ao conteúdo dos dados, mas também à **existência da informação**.

## Exemplos:

- Saber que uma pesquisa foi realizada pode ser mais revelador que seu resultado.
- O conhecimento de que houve perseguição por uma agência pode ser mais crítico que os detalhes dessa perseguição.

## Ocultação de Recursos:

- Organizações podem esconder sua infraestrutura para evitar abusos ou exposição.

*Mecanismos de controle de acesso podem ocultar tanto dados quanto sua própria existência.*

# VeraCrypt: Criptografia de Disco

**VeraCrypt** é uma ferramenta de código aberto para **criptografia de dados em disco**, derivada do TrueCrypt.

## Principais Características:

- Criptografia em tempo real (*on-the-fly encryption*);
- Suporte a algoritmos robustos: AES, Serpent, Twofish, e combinações;
- Possibilidade de criar volumes ocultos para negar a existência dos dados (*plausible deniability*);
- Funciona em Windows, Linux e macOS.

## Casos de Uso:

- Proteger HDs externos, pendrives e partições do sistema;
- Garantir confidencialidade em notebooks e dispositivos móveis;
- Armazenar backups criptografados de forma segura.

Link: Caso do uso TrueCrypt

## Proteção de Dados Médicos com Criptografia

- **Cenário Completo:**

- Rede de 5 hospitais compartilhando exames de imagem (raio-X, ressonâncias)
- Dados classificados como máxima sensibilidade

- **Solução Multicamadas:**

1. Criptografia AES-256 em repouso (armazenamento)
2. TLS 1.3 com certificados X.509 em trânsito
3. Chaves protegidas por HSM (Hardware Security Module)

- **Exemplo Detalhado:** Fluxo de um laudo de câncer de mama:

Etapa	Proteção Aplicada
Digitação médica	Criptografia de disco BitLocker
Transmissão de dados	Tunelamento VPN + TLS
Armazenamento nuvem	Criptografia e compliance

## Hierarquia de Acesso às Notas

- **Arquitetura:** Controle por papéis (RBAC) com políticas granulares

Perfil	Próprias Notas	Notas da Turma	Todas Notas
Aluno	✓	–	–
Professor	✓	✓	–
Coordenação	✓	✓	✓

- **Exemplo Prático:**

- Aluno João visualiza apenas suas notas em Álgebra
- Professora Maria vê todas as notas da turma de Cálculo II
- Coordenador Pedro acessa boletins de todos os alunos do curso

# Integridade

- **Integridade dos dados:** Garante que informações e programas sejam modificados apenas de maneira autorizada e específica.
- **Integridade do sistema:** Garante que o sistema desempenhe sua função prevista de forma íntegra, livre de manipulações não autorizadas, intencionais ou acidentais.

# Integridade: Confiança nos Dados

**Integridade** refere-se à **confiabilidade dos dados ou recursos**, geralmente no sentido de prevenir modificações indevidas ou não autorizadas.

## Dimensões da Integridade:

- **Integridade dos dados:** garante que o conteúdo da informação não foi alterado indevidamente.
- **Integridade da origem (autenticidade):** assegura que a fonte dos dados é confiável e legítima.

A integridade da origem afeta a **credibilidade**, que é fundamental para o funcionamento adequado de um sistema.

## Exemplo:

Um jornal publica uma informação obtida por vazamento da Casa Branca, mas atribui a fonte incorretamente.

⇒ **Integridade dos dados** preservada; **integridade da origem** comprometida.

# Mecanismos de Integridade

Os mecanismos de integridade dividem-se em duas classes: **mecanismos de prevenção** e **mecanismos de detecção**.

**Mecanismos de prevenção** mantêm a integridade dos dados bloqueando tentativas não autorizadas de alterar os dados ou modificá-los de formas não autorizadas.

Distinção importante:

- Tentativa de alterar dados sem autorização (ex: invasor tentando modificar dados).
- Tentativa de alterar dados de formas não autorizadas por usuários autorizados (ex: contador desviando dinheiro para conta no exterior).

**Mecanismos de detecção** indicam que a integridade dos dados foi comprometida.

- Analisar eventos do sistema (ações de usuários ou do sistema) para detectar problemas.
- Analisar os próprios dados para verificar se as restrições esperadas ainda são válidas.

Podem indicar a causa específica da violação ou apenas reportar que o arquivo está corrompido.

# Integridade vs. Confidencialidade

Trabalhar com integridade é muito diferente de trabalhar com confidencialidade.

Na confidencialidade, os dados ou foram comprometidos ou não.

Já a integridade inclui tanto a **correção** quanto a **confiabilidade** dos dados.

Fatores que afetam a integridade dos dados:

- A origem dos dados (como e de quem foram obtidos).
- O quão bem os dados foram protegidos antes de chegar à máquina atual.
- O quão bem os dados são protegidos na máquina atual.

Avaliar a integridade é difícil, pois depende de suposições sobre a origem dos dados e a confiança nessa origem — aspectos fundamentais de segurança que muitas vezes são negligenciados.

# Verificando a Integridade de Arquivos Baixados

Ao baixar arquivos da internet, é comum verificar sua integridade para garantir que:

- O arquivo não foi corrompido durante a transferência.
- O arquivo não foi adulterado por um atacante.

**Como fazer isso?** Usando funções de hash criptográficas como:

- MD5 – não recomendado hoje em dia mas muito usado.
- SHA-256 – mais recomendado atualmente.

**Exemplo no terminal:**

```
$ sha256sum arquivo.iso  
$ md5sum arquivo.iso
```

Compare o valor obtido com o publicado pelo site oficial.

# Sistema de acadêmico com garantia de integridade: Como garantir

- **Registro Oficial**

- Cada professor registra notas com login único e seguro
- Toda alteração precisa ser aprovada pela coordenação
- Data e hora são registradas automaticamente

- **Proteção contra Alterações**

- Notas lançadas não podem ser apagadas
- Mudanças criam um novo registro (como versão de documento)
- Relatórios mensais são conferidos pela diretoria

## Exemplo Prático: Professor dá nota para um aluno

1. O sistema registra:

- Quem lançou: Professora Ana
- Quando: 15/03/2023 às 14:30
- Versão do registro: 001

2. Se precisar corrigir, cria-se um novo registro (versão 002)

# Aula 3 - Conceitos Básicos de Segurança da Informação

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação  
Universidade Federal do Rio de Janeiro  
[gabrielaquino@ic.ufrj.br](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 2025

# Disponibilidade

- Garante que os sistemas **funcionem** corretamente e que os serviços **não sejam negados** aos usuários autorizados.
- Não adianta ter dados íntegros e protegidos se o sistema está fora do ar.

# Disponibilidade em Segurança

**Disponibilidade** refere-se à capacidade de acessar a informação ou o recurso desejado quando necessário.

- É essencial para a confiabilidade e o funcionamento do sistema.
- Um sistema indisponível é tão inútil quanto um sistema inexistente.
- Ataques como **DoS (Denial of Service)** exploram falhas de disponibilidade.
- Projetos de sistema geralmente assumem um *modelo estatístico* de uso.
- Um atacante pode distorcer esse modelo (ex: tráfego de rede excessivo) para causar falhas.

*Se o sistema não foi projetado para o cenário de ataque, os mecanismos de proteção podem falhar.*

# Preocupação com Disponibilidade

Availability %	Availability (Nines)	Downtime per Year	Downtime per Month	Downtime per Week	Downtime per Day
90%	One nine	36.53 days	73.05 hours	16.80 hours	2.40 hours
99%	Two nines	3.65 days	7.31 hours	1.68 hours	14.40 minutes
99.9%	Three nines	8.77 hours	43.83 minutes	10.08 minutes	1.44 minutes
99.99%	Four nines	52.60 minutes	4.38 minutes	1.01 minutes	8.64 seconds
99.999%	Five nines	5.26 minutes	26.30 seconds	6.05 seconds	864.00 milliseconds
99.9999%	Six nines	31.56 seconds	2.63 seconds	604.80 milliseconds	86.40 milliseconds
99.99999%	Seven nines	3.16 seconds	262.98 milliseconds	60.48 milliseconds	8.64 milliseconds
99.999999%	Eight nines	315.58 milliseconds	26.30 milliseconds	6.05 milliseconds	864.00 microseconds
99.9999999%	Nine nines	31.56 milliseconds	2.63 milliseconds	604.80 microseconds	86.40 microseconds

Link: Referência - Oracle

# De onde vêm os dados de disponibilidade?

Seus cálculos de disponibilidade serão tão bons quanto os dados neles inseridos.

Precisamos de dados sobre interrupções e tempo de inatividade.

Os dados de tempo de serviço e tempo de inatividade podem ser coletados de diversas fontes, incluindo:

- Teste de ping
- Software de monitoramento
- Tickets de suporte técnico / call center
- Relatórios de TI quando ocorrem incidentes de interrupção
- Sistemas de Gestão de Informações e Eventos de Segurança (SIEMs)
- Análise de IA de arquivos de log e outras entradas e saídas do sistema

# Disponibilidade - Exemplo de Ataque DDoS

## O Problema:

- **Cenário:** Sistema de matrículas online da universidade
- **Quando:** Primeiro dia do período de matrículas
- **Sintoma:** Sistema extremamente lento ou fora do ar

## O Que Aconteceu?

- Ataque DDoS (sobrecarga de acessos falsos)
- +100.000 solicitações por segundo de bots
- Servidores não conseguiram atender alunos reais

## Soluções de prevenção e recuperação:

### • Prevenção:

- Filtro contra tráfego malicioso
- Limite de tentativas por IP

### • Recuperação:

- Servidores reserva automáticos
- Sistema prioritário para alunos cadastrados

# Dificuldade em Detectar Ataques de Negação de Serviço (DoS)

- Ataques que visam tornar recursos indisponíveis são chamados de **ataques de negação de serviço (DoS)**.
- São **difíceis de detectar**, pois é necessário distinguir entre:
  - Manipulação intencional de recursos ou do ambiente;
  - Padrões de uso incomuns, mas legítimos.
- Os modelos estatísticos usados para prever o uso normal:
  - Consideram eventos atípicos como parte da distribuição estatística;
  - Podem não identificar um ataque como anômalo.
- Em alguns cenários, o ataque pode nem parecer atípico.

Para detectar comportamentos estranhos na rede...

Precisamos ver os dados de funcionamento da rede

# O que é um fluxo (flow) no NetFlow?

Um *flow* é definido como uma sequência unidirecional de pacotes com propriedades comuns que passam por um dispositivo de rede. Esses fluxos coletados são exportados para um dispositivo externo, o *NetFlow collector*.

Um registro de fluxo inclui:

- Endereços IP, Contagem de pacotes e bytes
- *Timestamps*, Tipo de Serviço (ToS)
- Portas de aplicação, Interfaces de entrada e saída

Os dados exportados do NetFlow são usados para:

- Faturamento de ISPs
- Monitoramento e planejamento de capacidade
- Monitoramento e perfil de aplicações e usuários
- Análise de segurança
- Mineração de dados para fins de marketing

Link: [RFC 3954 - NetFlow versão 9](#)

# Dados da rede - O que é NetFlow?

- Protocolo de rede desenvolvido pela **Cisco Systems** para coletar metadados sobre o tráfego IP em roteadores, switches e hosts.
- Permite monitorar e obter conhecimento sobre o desempenho de aplicações e da rede.
- Fornece informações sobre:
  - Quantidade de tráfego;
  - Origem e destino do tráfego;
  - Caminhos utilizados.
- Estatísticas de fluxo ajudam no monitoramento, identificação de problemas e planejamento de upgrades.
- Link: [O que é o Netflow](#)
- Link: [Configurando o Netflow em roteadores CISCO](#)
- Link: [Softflowd no Ubuntu](#)

# Dados da rede - Como funciona o NetFlow

## Principais componentes:

### 1. Exportador de NetFlow:

- Agrega pacotes em fluxos e exporta registros via UDP.
- Identifica fluxos com base em IP, portas, protocolo e tipo de serviço.
- Exporta fluxos inativos ou encerrados (ex: flags TCP FIN/RST).

### 2. Dados da rede - Coletor NetFlow:

- Recebe registros agregados de exportadores.
- Pré-processa e armazena os dados.

### 3. Dados da rede - Analisador de NetFlow:

- Processa e analisa os registros coletados.
- Gera relatórios e alertas sobre largura de banda, padrões de tráfego e uso de aplicações.

# Dados da rede - Solução NFSEN / NFDUMP

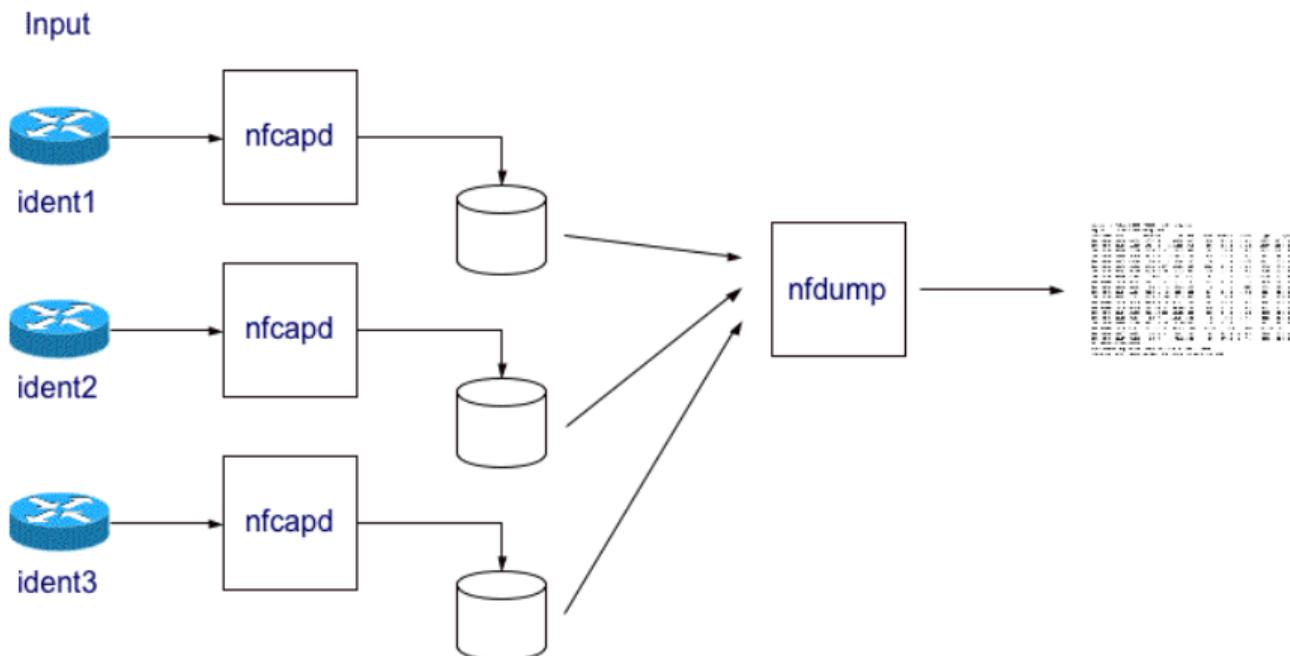
**NFDUMP:** Coleção de ferramentas para coleta e processamento de dados *NetFlow* via linha de comando. Faz parte do projeto **NfSen**.

- Link: [Github do projeto Nfdump](#)

## Principais ferramentas:

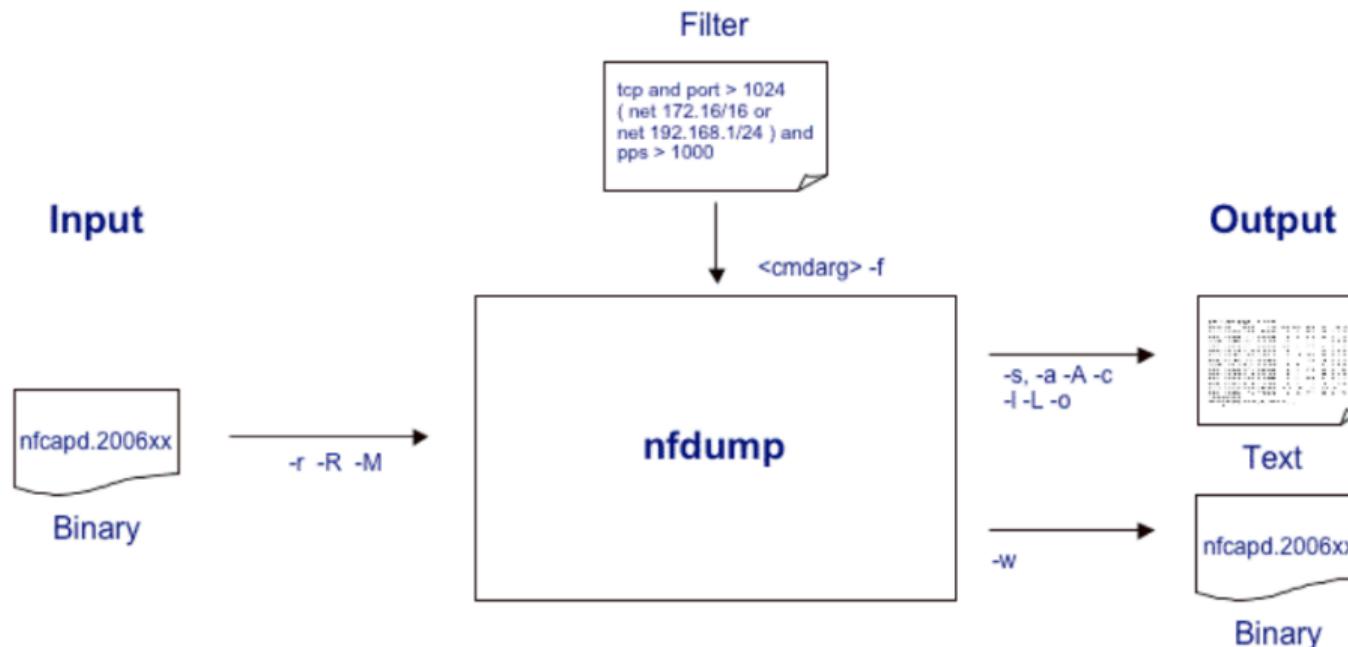
- **nfcapd** — Daemon que captura fluxos de rede (*NetFlow* v5, v7 e v9) e armazena em arquivos.
- **nfdump** — Lê e exibe dados *NetFlow* dos arquivos, semelhante ao *tcpdump*.
- **nfprofile** — Cria perfis de *NetFlow* com base em filtros.
- **nfreplay** — Reenvia dados de fluxo para outro host.
- **nfclean.pl** — Limpa dados antigos de forma periódica.
- **ft2nfdump** — Converte dados de outras ferramentas de fluxo para o formato *nfdump*.

# Dados da rede - Arquitetura NFDUMP



Link: [Ferramenta NFDUMP](#)

# Dados da rede - Como o NFDUMP funciona



Link: User Documentation nfdump e NfSen

**Front-end gráfico baseado na Web para as ferramentas de NetFlow do NFDUMP.**

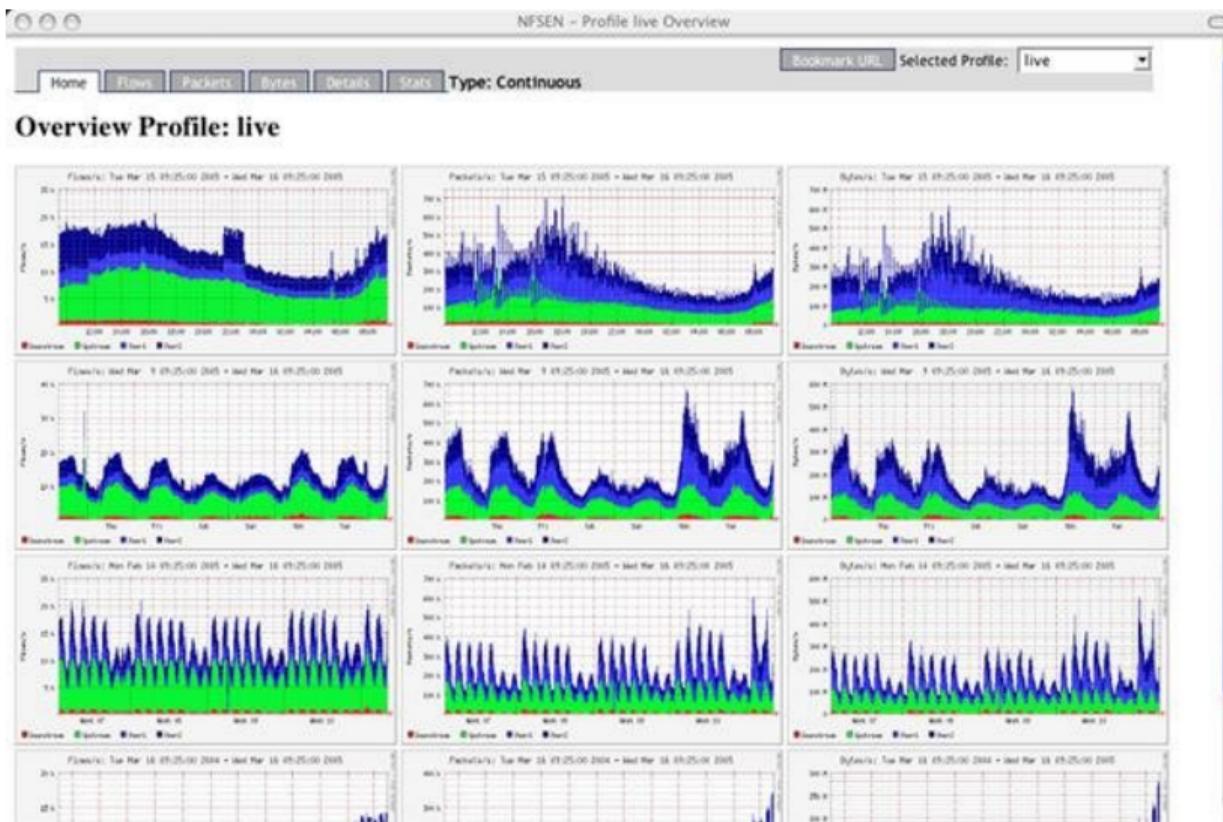
## Funcionalidades:

- Exibir e navegar facilmente pelos dados do NetFlow.
- Processar dados dentro de um período especificado.
- Criar histórico e perfis contínuos de tráfego.
- Definir alertas com base em várias condições.
- Desenvolver plugins personalizados para processar dados periodicamente.

*Disponível no SourceForge, sob licença BSD.*

- Link: [Sourceforge do NFSEN](#)

# Dados da rede - Front-End Nfsen



**ANÁLISE, DETECÇÃO E DESENVOLVIMENTO  
PARA PREVENÇÃO DE ANOMALIAS EM  
SISTEMAS DE COMUNICAÇÃO DE ALTO  
DESEMPENHO NO CIBERESPAÇO**

Layson Rodrigues da Costa (layson@cbpf.br)  
supervisora: Marita Maestrelli (marita@cbpf.br)

Jornada PCI-CBPF 2021/2022

Novembro de 2022

[Link: Apresentação no YouTube \(Começa em 13:21\)](#)  
[Link: Arquivo .pdf da apresentação](#)

# Aula 4 - Conceitos Básicos de Segurança da Informação

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação  
Universidade Federal do Rio de Janeiro  
[gabrielaquino@ic.ufrj.br](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 2025

# Três pilares da Segurança da informação

Os três pilares da segurança da informação são:

## Confidencialidade

**Definição:** Proteção contra acesso não autorizado

**Ameaça:** Vazamento de dados

**Exemplo:** Sigilo de processos judiciais

## Integridade

**Definição:** Prevenção de alterações indevidas

**Ameaça:** Fraude em registros

**Exemplo:** Sistemas para eleições

## Disponibilidade

**Definição:** Acesso contínuo aos sistemas

**Ameaça:** Ataques DDoS

**Exemplo:** Plataforma durante ataques ou crises

Exemplo dos três pilares em um sistema:

- Sigilo das transações (Confidencialidade)
- Manutenção dos saldos (Integridade)
- Operação 24/7 (Disponibilidade)

## Extensão dos três pilares: Autenticidade e Não-Repúdio

- **Autenticidade e não-repúdio:** Garantia da origem legítima das informações e impossibilidade de negar participação em uma transação.
  - **Autenticidade:** Propriedade que garante que a origem de uma informação é legítima e verificável.
  - **Não-repúdio:** Capacidade de provar que uma ação ocorreu e que sua origem não pode ser negada pelas partes envolvidas.

## Como ter autenticidade mas não ter não-repúdio? - Exemplo 1

**Cenário:** Um usuário preenche um formulário em papel, marcando opções com um “X” e assinando no final.

**Autenticidade:** A assinatura garante que o formulário foi realmente assinado pelo usuário — **autenticidade**.

**Não-Repúdio:** Posteriormente, o usuário alega: “Eu não marquei estas opções”. Como não há registro eletrônico ou prova criptográfica do preenchimento das opções, não é possível contestar a alegação — **não-repúdio**.

**Como garantir não repúdio:**

- Uso de formulários digitais com marcação eletrônica vinculada à assinatura digital.
- Registro seguro de cada ação com timestamp, de forma auditável.

## Como ter autenticidade sem ter não-repúdio? - Exemplo 2

**Cenário:** Um funcionário envia um pedido de reembolso via e-mail corporativo.

**Autenticidade:** O e-mail vem de sua conta corporativa e o sistema confirma que ele é o remetente — **autenticidade**.

**Não-repúdio:** O funcionário depois alega: “Eu não enviei este pedido, alguém acessou minha conta”. Sem uma assinatura digital ou registro auditável adicional, a empresa não consegue provar de forma incontestável que ele realmente enviou o e-mail — **não-repúdio**.

**Como garantir não-repúdio:**

- Assinatura digital do e-mail ou do pedido.
- Logs seguros e auditáveis da ação com timestamp.

# Autenticidade e não-repúdio

## Autenticidade

Propriedade de ser genuíno, verificável e confiável. Envolve ter confiança na validade de uma transmissão, mensagem ou origem da mensagem. Verifica se os usuários são quem dizem ser e se cada entrada recebida veio de uma fonte confiável.

Exemplo:

- Documento manuscrito: comparação das características de escrita com amostras verificadas.
- Informação eletrônica: uso de assinatura digital com criptografia de chave pública para verificar autoria e, possivelmente, a integridade.

## Não-repúdio

Capacidade de provar que um evento ou ação ocorreu e qual foi sua origem, evitando que as partes envolvidas neguem posteriormente.

- Prova de entrega para o remetente.
- Prova de identidade do remetente para o destinatário.

## Exemplo: Assinatura Digital via ITI (ICP-Brasil)

**Cenário:** Assina digital de documento utilizando o assinador do ITI ([Link](#)) Usando a Infraestrutura de Chaves Públicas (ICP) ([ICP-Brasil](#)).

- Certificado digital vinculado à identidade do usuário.
- Chave privada usada para assinar digitalmente o documento.
- Registro auditável da assinatura.
- Verificação automática de alterações posteriores ao documento assinado.

**Autenticidade:** O sistema verifica que a assinatura foi feita pelo titular do certificado digital ou pela conta `gov.br` — **autenticidade**.

**Não-Repúdio:** Assinatura feita com a chave privada do usuário e registrada no site do ITI (Assinador ITI), o usuário **não pode negar posteriormente** que assinou o documento — **não repúdio**.

**Integridade:** Se alguém tentar alterar o documento após a assinatura, o sistema detecta que ele foi modificado, garantindo que o conteúdo original permanece inalterado.

# Autenticidade e Não-Repúdio

## Autenticidade

- Garante que a mensagem ou ação veio realmente de quem afirma tê-la realizado.
- Exemplos:
  - Login com senha ou biometria.
  - Certificado digital em sites HTTPS.
  - Assinatura digital para verificação do remetente.

## Não-Repúdio

- Impede que o autor de uma ação negue sua responsabilidade posteriormente.
- Exemplos:
  - Assinaturas digitais com certificado ICP-Brasil.
  - Registros assinados de transações financeiras.
  - Logs de auditoria assinados digitalmente.

# Extensão dos três pilares: Responsabilização (Accountability)

- **Definição de Accountability:** Segurança que exige que ações de uma entidade sejam rastreadas exclusivamente para ela.
- **Objetivos de Accountability:**
  - Suportar o **não repúdio**.
  - Prevenir ou desencorajar **comportamentos indesejados**
  - Promover **isolamento de falhas**.
  - **Detectar e prevenir de intrusões**.
  - Facilitar a **recuperação após ação** e apoio para ações legais.
- **Importância:**
  - Permite rastrear brechas de segurança até a parte responsável.
  - Registros de atividades são fundamentais em uma para análise forense.
  - Dá insumos para se resolver disputas.

# Extensão dos três pilares: Responsabilização (Accountability)

## Responsabilização (Accountability)

- Capacidade de atribuir ações a indivíduos ou entidades específicas.
- Capacidade de exigir que indivíduos respondam por suas ações.
- Envolve identificação, autenticação e registro de atividades.
- Inclui mecanismos de auditoria, rastreabilidade e sanções.
- Permite investigação de incidentes de segurança.
- Exemplo: logs assinados digitalmente, políticas claras de uso.

Ou seja, precisamos lidar com quem fez o quê e tratar de como lidamos com quem fez o quê — ou seja, as consequências. Em termos técnicos: Você pode ter responsabilização, mas a responsabilização é a ação que responsabiliza alguém se sabe quem agiu. (ex: os logs mostram quem fez, mas nada é feito com isso)

# Exemplo de Responsabilização no Linux/Unix

**Ferramenta:** auditd (Auditing Daemon)

**Objetivo:** Capturar e registrar ações de usuários no sistema, garantindo rastreabilidade e possibilidade de responsabilização.

**Como funciona:**

- Logs de acesso localizados em `/var/log/audit/audit.log` ou `/var/log/secure`.
- Registram login/logout, comandos executados e alterações em arquivos críticos.
- Cada ação é associada a um UID, horário e tipo de operação.

**Exemplo prático:**

- Um usuário cria ou modifica um arquivo no sistema.
- auditd registra o evento com:
  - Identificação do usuário (UID)
  - Hora e data da ação
  - Comando executado ou arquivo alterado

**Uso:** Auditoria de segurança, investigação de incidentes, cumprimento de políticas corporativas e rastreabilidade legal.

# Auditd - Adicionar regra

Para monitorar um arquivo test\_audit.txt usando a tag test\_aula:

## Comandos

```
sudo auditd  
# Adiciona a regra para auditoria  
sudo auditctl -w ~/test_audit.txt -p wa -k test_aula  
  
# Verifica as regras ativas  
sudo auditctl -l  
  
# Executa alguma ação no arquivo...  
bash -c "echo 'linha de teste' >> /home/$USER/teste_audit.txt"  
  
# Consulta eventos pela tag  
sudo ausearch -k test_aula  
sudo ausearch -k test_aula --format text
```

## Auditd - Explicando os parâmetros

- auditctl — ferramenta para gerenciar regras no auditd.
- -w — caminho do arquivo ou diretório a ser monitorado.
- -p — permissões a auditar:
  - r — leitura
  - w — escrita
  - x — execução
  - a — alteração de atributos
- -k — chave (tag) para identificar os eventos, usada no ausearch.
- ausearch -k — busca no log do audit por eventos que contenham a chave especificada.

# Auditd - Interpretando a saída do ausearch

Ao executar:

```
ausearch -k test_aula
```

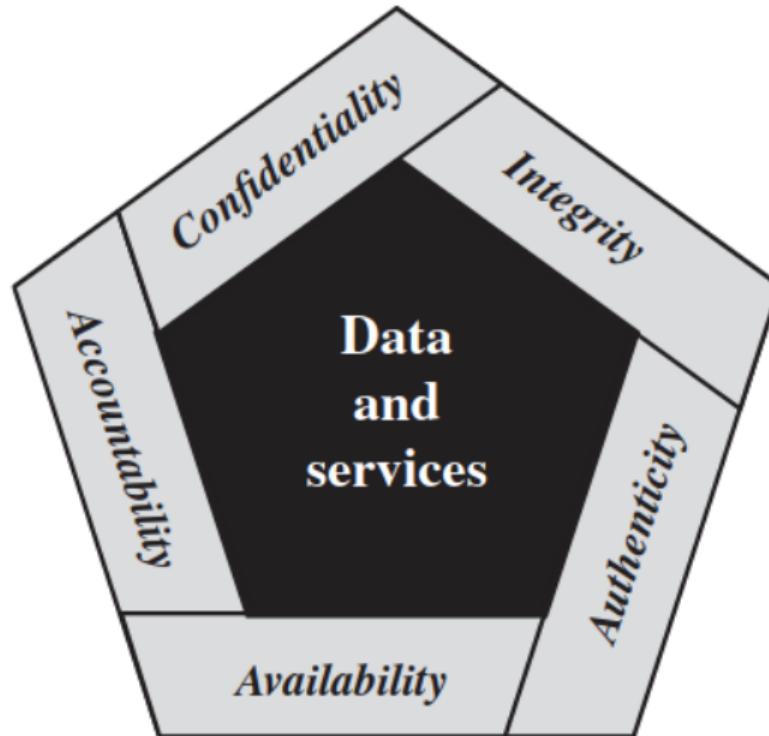
Na saída, é possível observar:

## Exemplo de trecho relevante

```
type=SYSCALL msg=audit(1692000000.123:420): \
    arch=c000003e syscall=257 success=yes \
    exit=3 a0=... auid=... uid=... comm="bash" exe="/usr/bin/bash" ...
    proctitle=6261736(...)7874
```

- syscall=257 corresponde à chamada de sistema openat, usada para abrir/criar arquivos.
- proctitle é exibido em hexadecimal. Ao decodificar 62617368..., obtemos: bash -c "echo a >> test\_audit.txt".
- Veja o usuário em *getent passwd "AUID"*

# Os 5 elementos da Segurança da Informação



**Figure 1.1 Essential Network and Computer Security Requirements**

# Aula 5 - Conceitos de criptografia e Criptografia Simétrica

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação  
Universidade Federal do Rio de Janeiro  
[gabrielaquino@ic.ufrj.br](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 2025

**Os sistemas criptográficos são caracterizados em três dimensões:**

**1. Tipo de operações:**

- **Substituição:** cada elemento do texto claro é mapeado em outro elemento.
- **Transposição:** os elementos do texto claro são rearranjados.
- A maioria dos sistemas combina ambos em várias etapas (sistemas de produto).

**2. Número de chaves:**

- **Encriptação simétrica:** mesma chave para emissor e receptor.
- **Encriptação assimétrica:** chaves diferentes (pública/privada).

**3. Modo de processamento:**

- **Cifra de bloco:** processa blocos inteiros de elementos.
- **Cifra de fluxo:** processa elementos continuamente, um a um.

- Também chamada de **encriptação convencional ou de chave única**.
- Era o único tipo em uso antes da década de 1970, quando surgiu a encriptação por chave pública.
- Continua sendo o muito utilizado até hoje.

## Definições iniciais:

- Texto claro (*plaintext*): mensagem original.
- Texto cifrado (*ciphertext*): mensagem codificada.
- Cifração (ou encriptação): processo de transformar texto claro em texto cifrado.
- Decifração (ou decriptação): processo de recuperar o texto claro a partir do texto cifrado.

# Criptografia, Criptoanálise e Criptologia

- **Criptografia:** estudo e construção de esquemas de encriptação (as chamadas cifras ou sistemas criptográficos).
- **Criptoanálise:** técnicas para decifrar uma mensagem sem conhecimento prévio da chave ou do método usado (o famoso "quebrar o código").
- **Criptologia:** área mais ampla que engloba tanto a criptografia quanto a criptoanálise.

**Resumo:** Criptografia = criar códigos. Criptoanálise = quebrar códigos. Criptologia = o estudo de ambos.

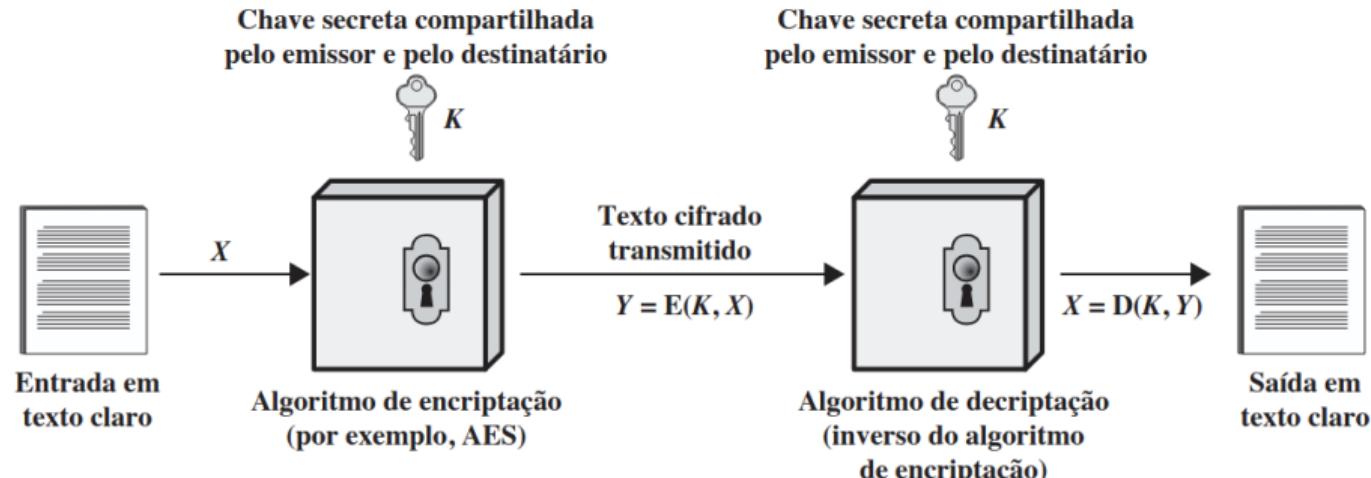
# Modelo de Cifra Simétrica

Um esquema de encriptação simétrica possui cinco componentes:

- **Texto claro:** mensagem original e inteligível, entrada do algoritmo.
- **Algoritmo de encriptação:** aplica substituições e transformações no texto claro.
- **Chave secreta:** valor independente do texto claro e do algoritmo; define as transformações realizadas.
- **Texto cifrado:** saída do processo de encriptação, com aparência aleatória e ininteligível.
- **Algoritmo de decriptação:** operação inversa à encriptação, recupera o texto claro a partir do texto cifrado e da chave.

# Esquema de Encriptação Simétrica

**Figura 2.1** Modelo simplificado da encriptação simétrica.



**Fluxo:** Texto Claro  $\xrightarrow{\text{Encriptação} + \text{Chave}}$  Texto Cifrado  $\xrightarrow{\text{Decriptação} + \text{Chave}}$  Texto Claro

# Requisitos para o Uso Seguro da Encriptação Simétrica

Para que a encriptação simétrica seja efetiva, dois requisitos fundamentais devem ser atendidos:

1. **Algoritmo forte:** mesmo conhecendo o algoritmo e tendo acesso a textos cifrados (com ou sem os respectivos textos claros), um oponente não deve ser capaz de descobrir a chave ou recuperar o texto claro.
2. **Chave protegida:** emissor e receptor devem compartilhar a chave secreta de forma segura e mantê-la em sigilo. Caso a chave seja comprometida, toda a comunicação associada poderá ser lida.

# Segurança na Encriptação Simétrica

- É impraticável decriptar uma mensagem apenas com o texto cifrado e o conhecimento do algoritmo.
- O segredo deve estar somente na **chave**, não no algoritmo.
- Essa característica torna a encriptação simétrica viável para uso generalizado.
- Como o algoritmo pode ser público, fabricantes desenvolvem **chips de baixo custo** com encriptação embutida, hoje comuns em diversos produtos.
- Portanto, o principal desafio de segurança é **manter o sigilo da chave**.

# Esquema de Encriptação Simétrica

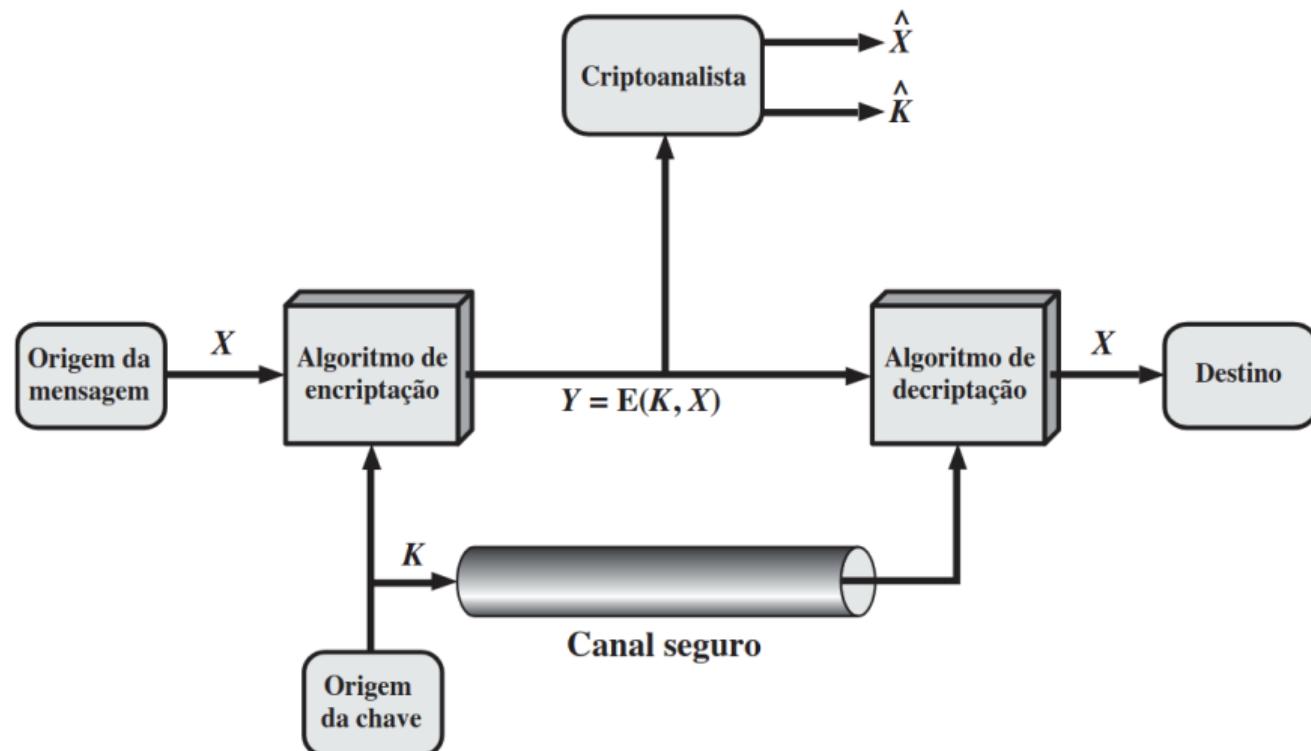
- Texto claro:  $X = [X_1, X_2, \dots, X_M]$
- Chave de encriptação:  $K = [K_1, K_2, \dots, K_J]$
- Texto cifrado:  $Y = E(K, X)$
- Decriptação:  $X = D(K, Y)$
- A chave  $K$  deve ser compartilhada com segurança entre origem e destino (diretamente ou por terceiro confiável).

## Visão do Oponente

- Oponente observa  $Y$ , mas não conhece  $K$ .
- Supõe-se que conhece os algoritmos  $E$  e  $D$ .
- Dois objetivos possíveis:
  - Recuperar o texto claro  $X$ , obtendo uma estimativa  $\hat{X}$ .
  - Recuperar a chave  $K$ , obtendo uma estimativa  $\hat{K}$  e assim ler mensagens futuras.

# Modelo de criptossistema simétrico

Figura 2.2 Modelo de criptossistema simétrico.



# Criptoanálise e Ataque por Força Bruta

O objetivo de um ataque é **recuperar a chave em uso**, não apenas o texto claro.

## Criptoanálise

- Explora a natureza do algoritmo de encriptação.
- Pode usar conhecimento prévio sobre o texto claro ou pares texto claro–cifrado.
- Busca deduzir a chave ou recuperar mensagens específicas.

## Ataque por Força Bruta

- Testa todas as chaves possíveis no texto cifrado.
- Em média, é necessário tentar metade do espaço de chaves.
- Garante sucesso se houver tempo e poder computacional suficiente.

# Os tipos de Ataques à criptografia

A forma de atacar depende da **quantidade de informação disponível** para o atacante:

- **Apenas texto cifrado**: cenário mais difícil, exige análise estatística.
- **Algoritmo conhecido**: normalmente se assume que o adversário sabe qual cifra está sendo usada.
- **Ataque por força bruta**: testar todas as chaves possíveis.
  - Impraticável se o espaço de chaves for muito grande.
- **Conhecimento do tipo de texto claro**: facilita a análise (ex.: textos em inglês/francês, arquivos EXE, código fonte, planilhas contábeis).

# Ataque de Texto Claro Conhecido

O ataque apenas com texto cifrado é o mais fácil de ser defendido, pois o oponente tem a quantidade mínima de informação para trabalhar

**Caso:** O atacante possui uma ou mais mensagens em **texto claro** e suas correspondentes mensagens cifradas.

- Ele pode explorar padrões previsíveis no texto claro.
  - Arquivos com cabeçalhos fixos (ex. PDFs ou logs).
  - Mensagens financeiras ou contratos eletrônicos com banners ou formatos padronizados.
  - Protocolos de rede com campos fixos (ex.: pacotes com cabeçalhos conhecidos).
- Com essas informações, é possível **deduzir a chave** ou reduzir significativamente o espaço de chaves a ser testado.
- Normalmente, ataques de texto claro conhecido são mais efetivos contra cifras fracas ou mal projetadas.

# Ataque de Palavra Provável e Texto Claro Escolhido

## Ataque de Palavra Provável:

- Variante do ataque de texto claro conhecido.
- O atacante possui conhecimento parcial de partes da mensagem ou de palavras específicas.
- Exemplos:
  - Arquivos de contabilidade com cabeçalhos padronizados ou palavras-chave em posições fixas.
  - Código fonte com notas de direitos autorais em posições específicas.
- Permite ao atacante reduzir o espaço de chaves a testar ou deduzir padrões na cifra.

## Ataque de Texto Claro Escolhido:

- O atacante consegue que a origem encripte mensagens de sua escolha.
- Pode introduzir padrões específicos que ajudem a revelar a estrutura da chave.

- Baseia-se em limitações computacionais ou econômicas do atacante.
- Dois critérios principais:
  - **Custo:** quebrar a cifra é mais caro que o valor da informação.
  - **Tempo:** quebrar a cifra leva mais tempo do que a vida útil da informação.
- Objetivo: tornar a decifração inviável na prática, mesmo que teoricamente possível.
- Um esquema de encriptação é considerado computacionalmente seguro se um desses dois critérios for atendido.
  - **Problema:** é muito difícil estimar a quantidade de esforço exigido para criptoanalisar textos cifrados com sucesso.
- Ciptoanálise para esquemas de encriptação simétricos são projetadas para explorar o fato de que rastros da estrutura ou do padrão do texto claro podem sobreviver à encriptação e ser discerníveis no texto cifrado

# Cifra de César

**Definição:** Uma cifra de substituição simples usada por Júlio César. Cada letra do alfabeto é substituída por aquela que está **três posições à frente**.

## Exemplo:

- Texto claro: meet me after the toga party
- Texto cifrado: PHHW PH DIWHU WKH WRJD SDUWB

## Alfabeto:

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

**Observação:** O alfabeto é circular, ou seja, após Z, volta-se para A.

# Tabela cifra de César

CHAVE	PHHW PH DIWHU WKH WRJD SDUWB
1	oggv og chvgt vjg vqic rctva
2	nffu nf bgufs uif uphb qbsuz
3	meet me after the toga party
4	ldds ld zesdq sgd snfz ozqsx
5	kccr kc ydrcc rfc rmey nyprw
6	jbbq jb xcqbo qeb qldx mxoqv
7	iaap ia wbpan pda pkcw lwnpu
8	hzzo hz vaozm ocz ojbv kvmot
9	gyyn gy uznyl nby niau julns
10	fxxm fx tymxk max mhzt itkmr
11	ewwl ew sxlwj lzw lgys hsjlq
12	dvvk dv rwkvi kyv kfxr grikp
13	cuuj cu qvjuh jxu jewq fqhjo
14	btti bt puitg iwt idvp epgin
15	assh as othsf hvs hcuo dofhm
16	zrrg zr nsgre gur gbtn cnegl
17	yqqf yq mrfqd ftq fasm bmdfk
18	xppe xp lqepc esp ezrl alcej
19	wood wo kpdob dro dyqk zkBDI
20	vnnC vn jocna cqn cxpj yjach
21	ummb um inbmz bpm bwoi xizbg
22	tlla tl hmaly aol avnh whyaf
23	skkz sk glzkx znk zumg vgxxze
24	rjjy rj fkyjw ymj ytlf ufwyd
25	qiix qi ejxiv xli xske tevxc

# Algoritmo Matemático da Cifra de César

**Cifra de César com deslocamento fixo (3 posições):**

$$C = E(3, p) = (p + 3) \mod 26$$

**Cifra de César geral com deslocamento  $k$ :**

$$C = E(k, p) = (p + k) \mod 26, \quad k \in \{1, 2, \dots, 25\}$$

**Decifragem:**

$$p = D(k, C) = (C - k) \mod 26$$

**Observações:**

- Cada letra do alfabeto é mapeada numericamente de 0 a 25.
- O operador  $\mod 26$  garante que o alfabeto seja circular (após Z vem A).
- Qualquer deslocamento  $k$  no intervalo de 1 a 25 gera uma cifra válida.
- $p$  representa cada letra do texto claro.

# Exemplo de Ataque por Palavra Provável - Cifra de César ( $k=6$ )

## Texto cifrado:

*G sotng igyg k asg igyg wak loig vkxzu jk uazxg igyg*

## Texto claro (César, deslocamento $k = 6$ ):

*A minha casa e uma casa que fica perto de outra casa*

**Observação:** A palavra "casa" aparece repetida várias vezes no texto cifrado (igyg).

## Ataque por palavra provável:

- Reconhecemos padrões repetidos (igyg) que correspondem à palavra "casa".
- Com isso, podemos deduzir o deslocamento usado ( $k = 6$ ).
- Após determinar a chave, podemos decifrar o restante do texto.

**Conclusão:** Mesmo sem conhecer a chave inicialmente, palavras repetidas permitem descobrir o deslocamento da Cifra de César.

# Criptoanálise da Cifra de César por Força Bruta

**Situação:** Se soubermos que o texto cifrado é uma cifra de César, podemos usar força bruta para descobrir o texto claro.

## Estratégia:

- Testar todas as 25 chaves possíveis ( $k = 1 \dots 25$ ).
- Observar qual saída produz texto inteligível.

## Exemplo:

- Texto cifrado: PHHW PH DIWHU WKH WRJD SDUWB
- Testando todas as chaves, o texto claro correto aparece na terceira tentativa: meet me after the toga party

## Condições que tornam a força bruta viável:

1. Algoritmos de encriptação e decriptação são conhecidos.
2. Espaço de chaves pequeno (apenas 25).
3. Linguagem do texto claro conhecida e reconhecível.

# Limitações da Força Bruta em Algoritmos Modernos

## Situação em redes e criptografia moderna:

- Algoritmos de encriptação geralmente são conhecidos.
- O espaço de chaves é enorme, tornando a força bruta impraticável.
  - Exemplo: Triple DES com chave de 168 bits →  $2^{168} \approx 3,7 \times 10^{50}$  chaves possíveis.
- Reconhecimento do texto claro pode ser difícil se:
  - A linguagem do texto não for conhecida.
  - O arquivo estiver compactado ou abreviado (ex.: ZIP).

**Conclusão:** A força bruta é viável apenas em cifras com pequeno espaço de chaves ou quando o texto claro é facilmente reconhecível, como na Cifra de César.

# Cifras Monoalfabéticas

**Definição:** Uma cifra monoalfabética usa uma permutação completa do alfabeto como chave. Cada letra do texto claro é substituída por uma letra correspondente no alfabeto cifrado.

**Exemplo:**

<b>Texto claro</b>	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
<b>Chave</b>	Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

**Observações:**

- A tabela de substituição (linha do texto cifrado) é a **chave** do sistema.
- Espaço de chave enorme:  $26! \approx 4 \times 10^{26}$  possibilidades.
- Muito mais seguro que a cifra de César contra ataques por força bruta.
- Cada letra do texto claro mapeia para exatamente uma letra do texto cifrado.

**Conclusão:** Em cifras monoalfabéticas, a chave é a própria permutação do alfabeto, garantindo um espaço de chave muito grande e maior resistência a ataques simples.

# Resumo da Lógica das Cifras Monoalfabéticas e César

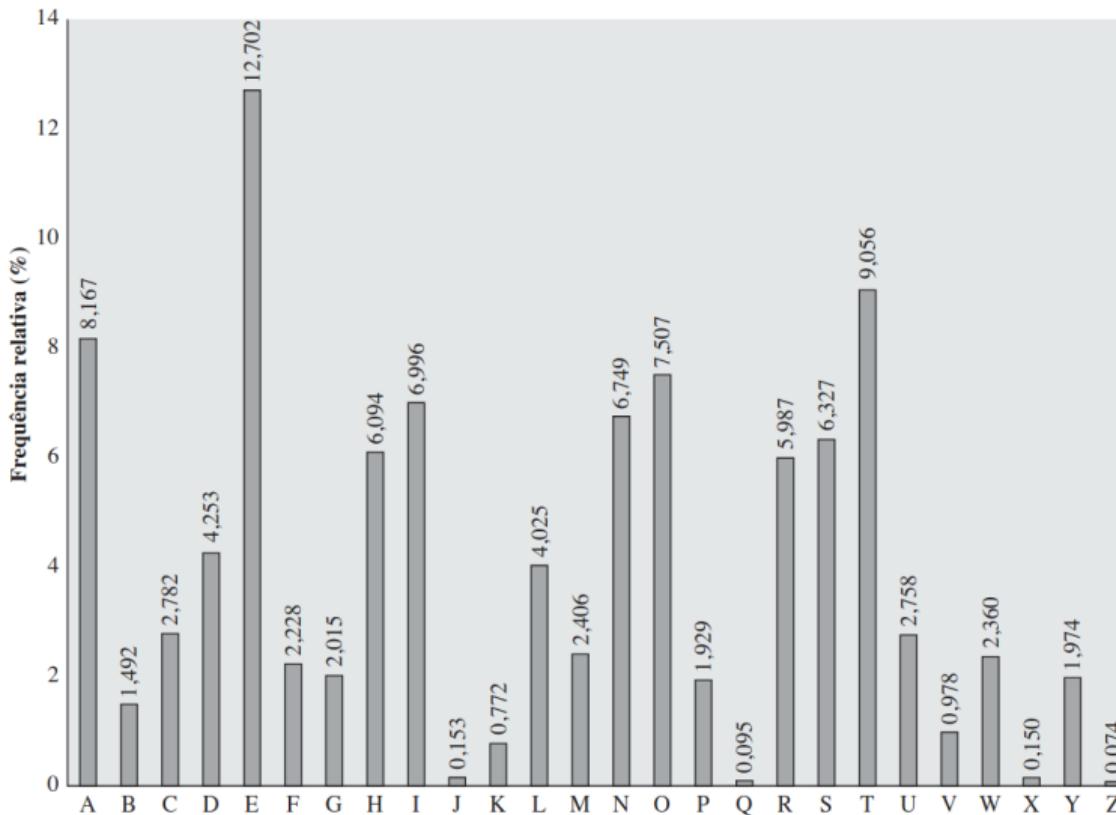
**O que é público:** O algoritmo, ou seja, como a substituição é feita. **O que é secreto:** A chave, que determina o mapeamento específico.

Característica	Cifra de César	Substituição Monoalfabética
Algoritmo	Substituir por deslocamento	Substituir por consulta a tabela
Chave	Número $K$ (ex: 3)	Tabela completa (ex: A→X, B→M, C→O, ...)
Espaço de chaves	Pequeno (25 chaves possíveis)	Enorme ( $26! \approx 4 \times 10^{26}$ )

**Observação:** A segurança de ambos depende do segredo da chave. Enquanto a Cifra de César é vulnerável à força bruta, a cifra monoalfabética é muito mais resistente, **mas ainda suscetível a ataques de análise de frequência**.

# O problema das cifras monoalfabéticas: a frequênciadas letras

**Figura 2.5** Frequência relativa de letras no texto em inglês.



# Cifras Monoalfabéticas com Homófonos

## Problema das cifras monoalfabéticas:

- Frequências de letras do alfabeto original permanecem no texto cifrado.
- Isso facilita ataques de análise de frequência.

## Solução: Homófonos

- Atribuir vários símbolos diferentes para a mesma letra do texto claro.
- Exemplo: a letra E poderia ser codificada como 16, 74, 35 ou 21, usados aleatoriamente ou em rodízio.
- Se o número de homófonos for proporcional à frequência da letra, a informação de frequência única é praticamente eliminada.

## Observações:

- Gauss acreditava ter criado uma cifra indecifrável usando homófonos.
- Mesmo com homófonos, padrões de múltiplas letras (digramas, trigrama) ainda podem ser explorados na criptoanálise.
- Portanto, ataques baseados em análise estatística ainda são possíveis.

# Importância dos Diagramas na Criptoanálise

## Por que diagramas importam?

- Análise de diagramas detecta **pares de letras** que aparecem frequentemente em um idioma.
- Mesmo cifras monoalfabéticas com homófonos ainda deixam padrões de diagramas visíveis.
- Cada idioma possui diagramas e trigramas comuns; explorá-los ajuda a quebrar cifras.

**Exemplos de diagramas comuns em português:** DE, ES, EN, NT, RE, RA, AR, OS, TE, CO

## Exemplo prático:

- Texto cifrado com substituição monoalfabética: símbolos %, &, \$# aparecem com frequência.
- Suspeita: % = E, & = A.
- Sequências %& e &% reforçam os diagramas EA e AE.
- Sequência \$%# sugere "NTE", como em "mente", "gente", "frente".
- Padrões de diagramas ajudam a deduzir a correspondência de símbolos e letras.

# Cifra Playfair

## Características principais:

- Cifra de múltiplas letras (substitui digramas, não letras isoladas).
- Cada par de letras no texto claro é convertido em um digrama no texto cifrado.
- Baseada em uma matriz  $5 \times 5$  de letras.

## Construção da matriz:

- Preencher a matriz com as letras da palavra-chave (sem duplicatas) da esquerda para a direita e de cima para baixo.
- Completar a matriz com as letras restantes do alfabeto.
- As letras I e J são consideradas uma só.

## Resumo do funcionamento:

- O texto claro é dividido em digramas (pares de letras).
- Cada digrama é substituído por outro digrama segundo regras da matriz.
- Essa abordagem dificulta a análise por frequência de letras individuais.

## Cifra playfair - Chave "monarchy"

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

# Cifra Playfair - Regras de Encriptação

**Criptografia por digramas:** O texto claro é encriptado duas letras de cada vez.

**Regras:**

- Letras repetidas em um par:** Inserir uma letra de preenchimento (ex.: 'x').  
Exemplo: balloon → ba lx lo on.
- Letras na mesma linha da matriz:** Substituir cada letra pela à direita, de forma rotativa. Exemplo: ar → RM.
- Letras na mesma coluna da matriz:** Substituir cada letra pela abaixo, de forma rotativa. Exemplo: mu → CM.
- Caso geral (retângulo):** Cada letra do par é substituída pela letra na sua linha e na coluna da outra letra do par. Exemplo: hs → BP, ea → IM (ou JM).

# Cifra Playfair - Contexto e Segurança

## Avanços sobre cifras monoalfabéticas:

- Enquanto uma cifra monoalfabética lida com 26 letras, a Playfair trabalha com  $26 \times 26 = 676$  digramas.
- A análise de frequência torna-se muito mais difícil, pois os padrões de digramas são menos evidentes.
- Foi considerada indecifrável por muito tempo.

## Uso histórico:

- Sistema de campo padrão do Exército britânico na Primeira Guerra Mundial.
- Ainda usada pelo Exército dos EUA e forças aliadas na Segunda Guerra Mundial.

## Limitações:

- Apesar da aparente segurança, a Playfair ainda deixa rastros da estrutura da linguagem do texto claro.
- Algumas centenas de letras de texto cifrado geralmente são suficientes para quebrá-la.

# Cifras Polialfabéticas e Vigenère

## Cifras polialfabéticas:

- Melhoram as cifras monoalfabéticas usando diferentes substituições ao longo da mensagem.
- Um conjunto de regras de substituição monoalfabéticas é utilizado.
- Uma chave define qual regra é aplicada a cada posição da mensagem.

## Cifra de Vigenère:

- Uma das mais conhecidas e simples cifras polialfabéticas.
- Utiliza 26 cifras de César com deslocamentos de 0 a 25.
- Texto cifrado  $C = C_0, C_1, \dots, C_{n-1}$  é calculado como:

$$C_i = (p_i + k_j) \mod 26$$

onde  $p_i$  é a letra do texto claro e  $k_j$  é a letra da chave, repetindo a chave conforme necessário.

**Resumo:** Cada letra do texto claro é cifrada usando uma regra que varia ao longo da mensagem, definida pela chave.

# Cifra de Vigenère: Operação Detalhada

## Como funciona:

- Cada letra do texto claro é cifrada usando uma letra da chave correspondente.
- Para textos mais longos que a chave, a chave é repetida ciclicamente.

## Equação de encriptação:

$$C_i = (p_i + k_i \bmod m) \bmod 26$$

- $p_i$ : letra do texto claro na posição  $i$
- $k_i \bmod m$ : letra da chave correspondente, repetida quando necessário
- $m$ : comprimento da chave

## Equação de decriptação:

$$p_i = (C_i - k_i \bmod m) \bmod 26$$

**Comparação com César:** Cada caractere do texto claro é cifrado com uma cifra de César diferente, determinada pela letra da chave.

## Cifra de Vigenère - Exemplo

- **Palavra-chave:** "deceptive"
- **Mensagem:** "we are discovered save yourself"

chave:	deceptive																										
texto claro:	w	e	a	r	e	d	i	s	c	o	n	d	e	r	s	a	v	e	y	o	u	r	s	o	u	f	
texto cifrado:	Z	I	C	V	T	W	Q	N	G	R	Z	G	V	T	W	A	V	Z	H	C	Q	Y	G	L	M	G	J

Expresso numericamente, temos o seguinte resultado:

chave	3	4	2	4	15	19	8	21	4	3	4	2	4	15
texto claro	22	4	0	17	4	3	8	18	2	14	21	4	17	4
texto cifrado	25	8	2	21	19	22	16	13	6	17	25	6	21	19

# Vulnerabilidade da Cifra de Vigenère

**Ocultação da frequência de letras:** Cada letra do texto claro pode gerar múltiplas letras cifradas, dependendo da letra correspondente da chave, dificultando a análise de frequência simples.

**Informações ainda presentes:** Apesar da melhoria em relação à cifra Playfair, padrões do texto claro ainda podem aparecer. Por exemplo, sequências repetidas no texto cifrado podem indicar o comprimento da chave.

**Exemplo:** Um analista observa que a sequência “VTW” aparece duas vezes a uma distância de 9 caracteres. Isso sugere que a chave pode ter 3 ou 9 letras de extensão, embora a repetição possa ser casual.

**Ataque possível:** Se a mensagem for longa, múltiplas sequências repetidas permitem ao analista estimar o tamanho da chave. Esse é o princípio do ataque de Kasiski, usado para quebrar cifras polialfabéticas.

# Cifra de Vigenère com Auto-Chave

**Problema da repetição da chave:** A periodicidade da palavra-chave facilita ataques de criptoanálise.

**Solução proposta por Vigenère:** Usar uma palavra-chave concatenada ao próprio texto claro, formando uma *chave corrente* tão longa quanto a mensagem.

**Exemplo:**

- Chave: `deceptivewearediscoveredsav`
- Texto claro: `wearediscoveredsaveyourself`
- Texto cifrado: `ZICVTWQNGKZEIIGASXSTSLVVWLA`

**Observação:** Mesmo assim, o esquema ainda é vulnerável à criptoanálise estatística, pois a chave compartilha a distribuição de frequência do texto claro.

# Cifra de Vernam

**Ideia principal:** Usar uma palavra-chave tão longa quanto o texto claro, sem relação estatística com ele.

**Introdução:** Proposta em 1918 por *Gilbert Vernam*, engenheiro da AT&T. Opera sobre **dados binários** (bits), e não sobre letras.

**Definição:**

$$c_i = p_i \oplus k_i$$

$$p_i = c_i \oplus k_i$$

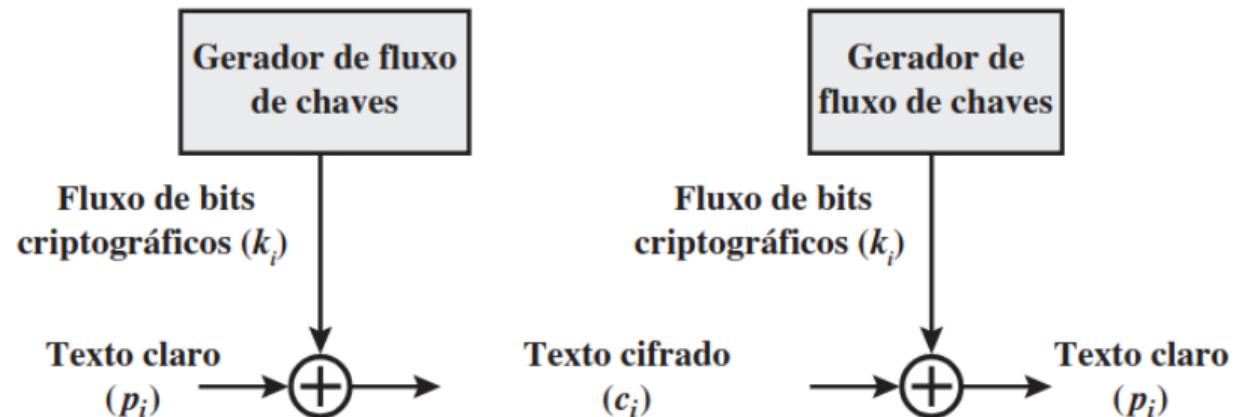
onde:

- $p_i$  = bit do texto claro na posição  $i$ ,
- $k_i$  = bit da chave na posição  $i$ ,
- $c_i$  = bit do texto cifrado na posição  $i$ ,
- $\oplus$  = operação *XOR* (OU-exclusivo).

**Observação:** Vernam propôs uma chave muito longa, mas ainda repetida. Com texto cifrado suficiente, o sistema pode ser quebrado com análise estatística e *texto claro provável*.

# Cifra de Vernam

**Figura 2.7** Cifra de Vernam.



## Melhoria proposta por Joseph Mauborgne (Exército dos EUA):

- Uso de uma **chave aleatória**, tão longa quanto a mensagem.
- A chave nunca é repetida.
- Cada mensagem exige uma nova chave, que deve ser descartada após o uso.

## Propriedades:

- O texto cifrado é totalmente aleatório.
- Não existe correlação estatística com o texto claro.
- **Inquebrável**: não há informação no texto cifrado que permita deduzir o texto claro sem a chave.

## Exemplo - One-time pad

ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUFPLUYTS

Agora mostramos duas decriptações diferentes usando duas chaves distintas:

texto cifrado: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUFPLUYTS

chave: *pxlmvmsydoфuyrvzwc tnleбnecvgdупahfzzlmnyih*

texto claro:<sup>\*</sup> mr mustard with the candlestick in the hall

texto cifrado: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUFPLUYTS

chave: *mfugpmiydgaxgoufhk111mhsqdqogtewbqfgyovuhwt*

texto claro:<sup>\*\*</sup> miss scarlet with the knife in the library

# One-Time Pad: Segurança e Limitações

## Segurança:

- A segurança decorre inteiramente da **aleatoriedade da chave**.
- Se a chave é verdadeiramente aleatória, o texto cifrado também será.
- Não existem padrões ou regularidades para um criptoanalista explorar.
- Representa a **cifra perfeita**: segurança completa em teoria.

## Limitações práticas:

1. Geração de grandes quantidades de números verdadeiramente aleatórios.
2. Distribuição e proteção das chaves — uma nova chave, do mesmo tamanho da mensagem, é necessária para cada comunicação.

## Segredo perfeito

Por causa dessas dificuldades, o one-time pad tem utilidade limitada, e aplicação principalmente para canais de pouca largura de banda que exigem segurança muito alta. O one-time pad é o único criptossistema que apresenta o que é conhecido como **segredo perfeito**.

# Aula 6 - Cifras de Fluxo

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação  
Universidade Federal do Rio de Janeiro  
[gabrielaquino@ic.ufrj.br](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 2025

# Princípio de Kerckhoffs

## Definição

O Princípio de Kerckhoffs, formulado por Auguste Kerckhoffs no século XIX, afirma que a segurança dos dados encriptados deve depender apenas da chave usada, mesmo que o método ou algoritmo seja de conhecimento público.

## Implicação

Esse princípio denuncia a falácia da *segurança pela obscuridade*, evidenciando que, se a segurança depende do segredo do método, então o método é falho.

# Cifras de fluxo

- Uma **técnica de substituição** é aquela em que as letras do texto claro são substituídas por outras letras, números ou símbolos. Se o texto claro for visto como uma sequência de bits, então a substituição envolve trocar padrões de bits de texto claro por padrões de bits de texto cifrado.
- **Cifra de fluxo:** Encripta um fluxo de dados digital um bit ou um byte por vez.
- Exemplos clássicos:
  - Vigenère autochaveada
  - Vernam

## Da aula passada: Cifra de Fluxo: Vernam (One-Time Pad)

- Cada bit do texto claro é combinado com um bit da chave usando **XOR**.
- Se a chave for tão longa quanto a mensagem e aleatória, temos o **One-Time Pad**, inquebrável.
  - É uma versão ideal da cifra de Vernam.
  - O fluxo de chaves ( $k_i$ ) tem o mesmo tamanho do fluxo de bits do texto claro ( $p_i$ ).
  - Se a chave for realmente aleatória, a cifra é **inquebrável**.
  - Limitação: o fluxo de chaves precisa ser **pré-distribuído** de forma segura.
  - Problema prático: inviável para grandes volumes de tráfego devido à logística de distribuição.
- Se a chave for curta e repetida, a segurança é comprometida.

# XOR: Base das Cifras de Fluxo

## Tabela-verdade da função XOR

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Onde  $S = A \oplus B$

## Exemplo de Encriptação e Decriptação

Para cifrar ( $c$ ) =  $TEXTO \oplus CHAVE$ :

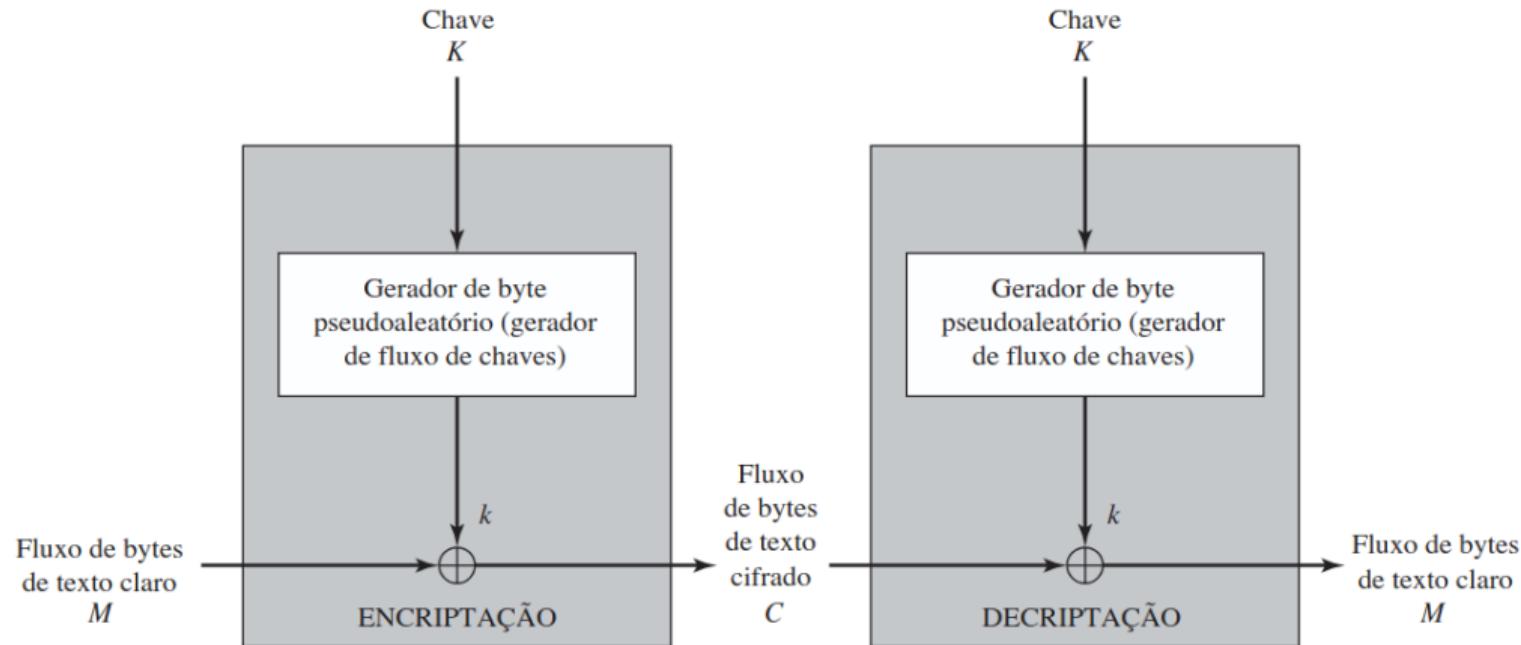
Texto claro (p)	1	0	1	1
Chave (k)	0	1	0	1
Cifrado (c)	1	1	1	0

Para decifrar ( $d$ ):  $CIFRADO \oplus CHAVE$ :

Cifrado (c)	1	1	1	0
Chave (k)	0	1	0	1
Texto claro (p)	1	0	1	1

# Cifra de fluxo

**Figura 7.7** Diagrama da cifra de fluxo.



- Uma cifra de fluxo típica encripta um byte de texto claro por vez, embora possa operar sobre um bit ou unidades maiores.
- Uma chave é inserida em um gerador de bits pseudoaleatórios que produz um fluxo de números de 8 bits aparentemente aleatórios.
- A saída do gerador, chamada *fluxo de chaves*, é combinada byte a byte com o texto claro usando a operação **OU exclusivo (XOR)** bit a bit.

## Cifra de fluxo

o byte de texto cifrado resultante será

$$\begin{array}{r} 11001100 \text{ texto claro} \\ \oplus \quad \underline{01101100} \text{ fluxo de chaves} \\ 10100000 \text{ texto cifrado} \end{array}$$

A decriptação requer o uso da mesma sequência pseudoaleatória:

$$\begin{array}{r} 10100000 \text{ texto cifrado} \\ \oplus \quad \underline{01101100} \text{ fluxo de chaves} \\ 11001100 \text{ texto claro} \end{array}$$

# Geradores de Fluxo em Cifras de Fluxo

- Por motivos práticos, o fluxo de bits é gerado por um **algoritmo** controlado por chave.
- O gerador deve produzir um fluxo **criptograficamente forte**.
  - Não deve ser possível prever partes futuras do fluxo a partir de partes já conhecidas.
- Os usuários compartilham apenas a **chave de geração**.
- Cada usuário pode então produzir localmente o mesmo fluxo de chaves.

## Semelhança com One-Time Pad:

- Uma cifra de fluxo também encripta um fluxo de bits de texto claro com um fluxo de chave.
- No one-time pad, o fluxo de chave é **verdadeiramente aleatório**.
- Na cifra de fluxo, o fluxo de chave é **pseudoaleatório**, gerado por um algoritmo.

## Considerações de projeto para cífras de fluxo:

1. A sequência de encriptação deverá ter um período grande?
2. O fluxo de chaves deverá se aproximar o máximo possível das propriedades de um fluxo de número aleatório verdadeiro?
3. A chave precisa ser suficientemente longa?

# Período de Repetição em Cifras de Fluxo

## Sequência de Encriptação:

- A sequência de bits gerada pelo gerador pseudoaleatório deve ter **um período longo**.
- O gerador produz um fluxo determinístico de bits que eventualmente se repete.
- Quanto maior o período, mais difícil é a criptoanálise.

## Relação com a cifra de Vigenère:

- Na Vigenère, uma palavra-chave curta se repete, facilitando ataques por análise de frequência.
- Na cifra de fluxo, um período maior no gerador pseudoaleatório reduz padrões repetitivos, aumentando a segurança.

## Requisitos para o fluxo de chaves em cifras de fluxo:

- O fluxo deve se aproximar de **um número aleatório verdadeiro**.
- Deve conter aproximadamente o mesmo número de 1s e 0s.
- Se o fluxo for tratado como bytes, os 256 valores possíveis devem aparecer com frequência aproximadamente igual.

## Impacto na segurança:

- Quanto mais aleatório o fluxo de chaves, mais aleatório será o texto cifrado.
- Isso dificulta a criptoanálise, pois padrões previsíveis são minimizados.

# Tamanho da Chave em Cifras de Fluxo

## Considerações sobre a chave:

- A saída do gerador de número pseudoaleatório depende da **chave de entrada**.
- Para proteção contra ataques de força bruta, a **chave deve ser suficientemente longa**.
- Assim como em cifras de bloco, recomenda-se **chaves de pelo menos 128 bits** com a tecnologia atual.

**Importância:** Uma chave curta compromete a segurança, mesmo que o gerador de fluxo seja forte.

# Comparação entre Cifras de Fluxo e de Bloco

## Segurança:

- Uma cifra de fluxo bem projetada pode ser tão segura quanto uma cifra de bloco com a mesma **tamanho de chave**.

## Vantagens das cifras de fluxo:

- Geralmente mais rápidas que cifras de bloco.
- Requerem menos código para implementação.
- Exemplo: RC4 pode ser implementado em poucas linhas.

# Comparação entre Cifras de Fluxo e de Bloco

## Atenção:

- Nos últimos anos, essa vantagem diminuiu com a introdução do AES, que é bastante eficiente em software.
- Técnicas de aceleração de hardware, como o AES Instruction Set da Intel, permitem executar uma rodada de encriptação/decriptação e geração de chave com grande velocidade.
- Ganhos de velocidade podem ser de cerca de uma ordem de grandeza em relação a implementações puramente em software.

# Vantagens e Riscos das Cifras de Bloco e Fluxo

## Cifras de Bloco:

- Permitem **reutilização de chaves** sem comprometer a segurança.

## Cifras de Fluxo:

- Se dois textos claros forem encriptados com a mesma chave, a criptoanálise torna-se trivial.
- O XOR dos dois textos cifrados produz o XOR dos textos claros originais.
- Vulnerável especialmente quando os textos claros possuem padrões conhecidos, como strings de texto, números de cartão de crédito ou outros fluxos de bytes estruturados.

# Escolha de Cifras: Fluxo vs. Bloco

## Cifras de Fluxo:

- Adequadas para **encriptação/decriptação contínua** de dados.
- Ex.: canais de comunicação de dados, links de navegador/Web.

## Cifras de Bloco:

- Indicadas para **blocos de dados** inteiros.
- Ex.: transferência de arquivos, e-mails, bancos de dados.

**Observação:** Ambos os tipos podem ser usados em praticamente qualquer aplicação, dependendo do projeto.

# Cifra de Fluxo RC4

## Descrição:

- Criada em 1987 por Ron Rivest para a RSA Security.
- Tamanho de chave **variável**, operações orientadas a **byte**.
- Baseada em uma **permutação aleatória**.

## Características:

- Período provavelmente maior que  $10^{100}$ .
- De 8 a 16 operações de máquina por byte de saída.
- Executa rapidamente em software.

## Hoje é considerado **inseguro**, mas já teve uso:

- SSL/TLS (comunicação navegador-servidor Web)
- WEP e WPA (padrão IEEE 802.11 LAN sem fio)

## Histórico:

- Mantido como segredo comercial até 1994.
- Postado na Internet na lista Cypherpunks em setembro de 1994.

# RC4: Inicialização e Operação

## Inicialização:

- Chave de tamanho variável: 1 a 256 bytes (8 a 2048 bits).
- Vetor de estado  $S$  de 256 bytes:  $S[0], S[1], \dots, S[255]$ .
- $S$  contém uma **permutação de todos os números de 0 a 255**.

## Operação (Encriptação/Decriptação):

- Um byte  $k$  é gerado a partir de  $S$  selecionando uma entrada de forma sistemática.
- Cada vez que um byte  $k$  é gerado, os elementos de  $S$  são **novamente permutados**.
- A saída  $k$  é usada para XOR com o byte do texto claro/cifrado:

$$c_i = p_i \oplus k, \quad p_i = c_i \oplus k$$

- $p_i$  i-ésimo bit (ou byte) do texto claro.
- $k$  byte (ou bit) gerado pelo gerador de fluxo da cifra RC4.
- $c_i$  i-ésimo byte (ou bit) do texto cifrado,

# Inicialização do Vetor S no RC4

- Defina o vetor de estado  $S$  com 256 elementos:

$$S[0] = 0, S[1] = 1, \dots, S[255] = 255$$

- Crie um vetor temporário  $T$  do mesmo tamanho que  $S$ .
- Preencha  $T$  usando a chave  $K$  de tamanho  $\text{keylen}$ :
  - Se  $\text{keylen} = 256$ , copie  $K$  diretamente para  $T$ .
  - Caso contrário, copie os primeiros  $\text{keylen}$  bytes de  $K$  para  $T$ , repetindo a chave tantas vezes quanto necessário até completar 256 elementos.

```
/* Inicialização */  
for i = 0 to 255 do  
    S[i] = i;  
    T[i] = K[i mod keylen];
```

## Exemplo - Inicialização do RC4 - Passo 1: Vetores S e T

**Chave:** 11111001 (1 byte)

**Vetor S:** Inicialmente com 256 elementos em ordem crescente:

$$S = [0, 1, 2, 3, \dots, 255]$$

**Vetor T:** Criado a partir da chave, repetida até ter 256 elementos:

$$T = [11111001, 11111001, 11111001, \dots] \text{ (256 elementos)}$$

**Resumo:**

- S contém todos os números de 0 a 255.
- T é preenchido sequencialmente com os bytes da chave, repetindo-a se necessário.

## RC4 - permutação

- Em seguida, usamos T para produzir a permutação inicial de S.
- Isso envolve começar com  $S[0]$  e ir até  $S[255]$ , e, para cada  $S[i]$ , trocar  $S[i]$  por outro byte em S, de acordo com um esquema ditado por  $T[i]$
- Como a única operação sobre S é uma troca, o único efeito é uma permutação. S, ainda, contém todos os números de 0 a 255.

```
/* Permutação inicial de S */  
j = 0;  
for i = 0 to 255 do  
    j = (j + S[i] + T[i]) mod 256;  
Swap (S[i], S[j]);
```

## RC4 - geração do fluxo

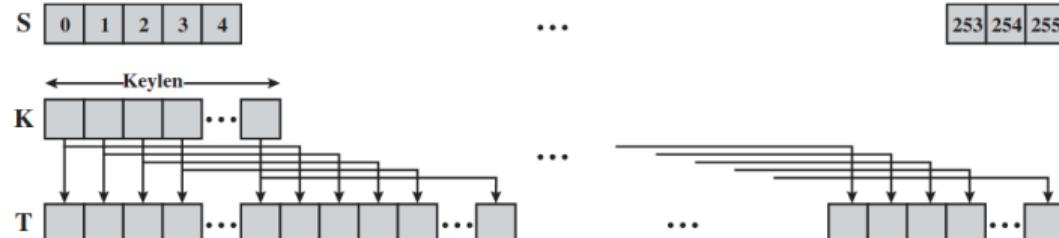
- Uma vez que o vetor S é inicializado, a chave de entrada não é mais usada.
- A geração de fluxo é percorrer todos os elementos de S[i] e, para cada S[i], trocar S[i] por outro byte em S de acordo com um esquema ditado pela configuração atual de S.
- Depois que S[255] é atingido, o processo continua, começando novamente em S[0]:

```
/* Geração de fluxo */
i, j = 0;
while (true)
    i = (i + 1) mod 256;
    j = (j + s[i]) mod 256;
    Swap (s[i], s[j]);
    t = (s[i] + s[j]) mod 256;
    k = s[t];
```

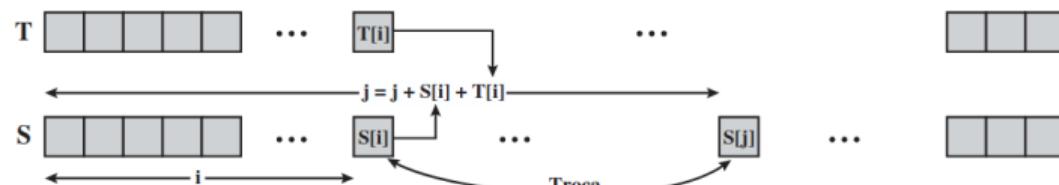
- Para encriptar, faça o XOR do valor k com o próximo byte do texto claro. Para decriptar, faça o XOR do valor k com o próximo byte do texto cifrado.

# RC4 - big picture

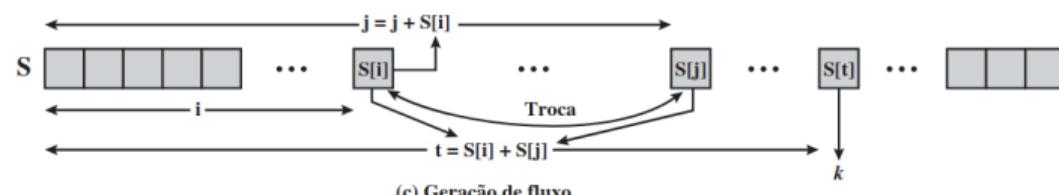
Figura 7.8 RC4.



(a) Estado inicial de S e T



(b) Permutação inicial de S



(c) Geração de fluxo

## Segurança do RC4:

- RC4 é resistente a ataques práticos se a chave for suficientemente longa (ex: 128 bits).
- Muitos estudos analisaram métodos de ataque, mas não são práticos para chaves de tamanho adequado.

## Problema no WEP:

- O protocolo WEP (para redes 802.11) mostrou-se vulnerável devido à forma de geração de chaves.
- O problema não está no RC4 em si, mas no uso incorreto da chave.
- O uso do WEP é desencorajado hoje em dia

# Aula 7 - Wired Equivalent Privacy (WEP)

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação  
Universidade Federal do Rio de Janeiro  
[gabrielaquino@ic.ufrj.br](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 2025

# Introdução ao WEP

- Nos primeiros cinco anos do padrão IEEE 802.11, apenas um método de segurança foi definido: o **Wired Equivalent Privacy (WEP)**.
- Muitas vezes, WEP foi identificado incorretamente como *Wireless Effective Privacy* e outras variantes.
- Com o aumento da popularidade das redes Wi-Fi no ano 2000, a comunidade criptográfica começou a analisar o WEP e rapidamente encontrou vulnerabilidades.
- Já em 2001, ferramentas para quebrar o WEP em pouco tempo estavam disponíveis na Internet.

# Objetivos do WEP segundo o IEEE 802.11 (1999)

- **Força Razoável:**
  - A segurança depende da dificuldade de descobrir a chave secreta por ataque de força bruta.
  - Relaciona-se ao tamanho da chave e à frequência de troca das chaves e do vetor de inicialização (IV).
- **Exportabilidade:**
  - Foi projetado de modo a facilitar a aprovação de exportação pelo Departamento de Comércio dos EUA.
  - No entanto, devido ao clima político e legal da época, não havia garantias de exportabilidade.
  - O padrão IEEE 802.11 especificava apenas o uso de chaves de 40 bits.
  - **Problema:** 40 bits é um tamanho muito pequeno para resistir a ataques de força bruta, o que explicava sua aceitação nas regras de exportação.
  - **Exemplo de ideia da época:** Se um banco fosse utilizar LAN sem fio, teria seu próprio protocolo de segurança rodando sobre o WEP.

# Objetivos do WEP segundo o IEEE 802.11 (1999)

- **Auto-Sincronização:**
  - O WEP é auto-sincronizável para cada mensagem
  - Propriedade importante em algoritmos de enlace de dados, onde a perda de pacotes pode ser alta.
  - Basicamente, isso significa que cada pacote deve ser encriptado separadamente, de forma que, dado um pacote e a chave, você tenha todas as informações necessárias para decriptá-lo.
  - Claramente, não se deseja uma situação em que a perda de um único pacote torne todos os pacotes seguintes indecifráveis.
- **Eficiência:** O algoritmo é eficiente, podendo ser implementado em **hardware ou software**.
- **Opcionalidade:** A implementação e o uso do WEP eram **opcionais** no padrão IEEE 802.11.

# Como o WEP foi "vendido" na época

- **Influência do Marketing:**
  - Na promoção do IEEE 802.11, a palavra "razoável" foi omitida
  - Na época WEP passou a ser descrito simplesmente como seguro.
  - Após a flexibilização das restrições de exportação, fabricantes criaram extensões não padronizadas com chaves de 104 bits
  - **Propaganda:** WEP era "extremamente" ou "absolutamente" seguro.
- **Efeito no Mercado** Essas extensões de chave foram incorporadas à especificação Wi-Fi e se tornaram o padrão da indústria em 1999. Na visão dos gestores de marketing, o WEP estava agora completamente seguro.
- **IMPORTANTE: Erro na Definição de Segurança:**
  - Aceitar o conceito de um nível "razoável" de segurança foi um erro.
  - Só existem dois tipos de segurança: forte ou nenhuma.
  - O padrão deveria: (i) OU ter incorporado uma solução realmente robusta OU deixado claro que a segurança deveria ser provida por outros meios (ex. VPN, HTTPS..).

## Críticas ao WEP

- Alguns críticos acusam os projetistas do padrão IEEE 802.11 original por criarem o WEP com fraquezas inerentes.
- No entanto, é importante lembrar que, na época, o WEP **não foi projetado para prover segurança em nível militar**.
- O objetivo era fornecer uma proteção semelhante à de uma rede cabeada: **difícil, mas não impossível de quebrar**.

# Uso do WEP

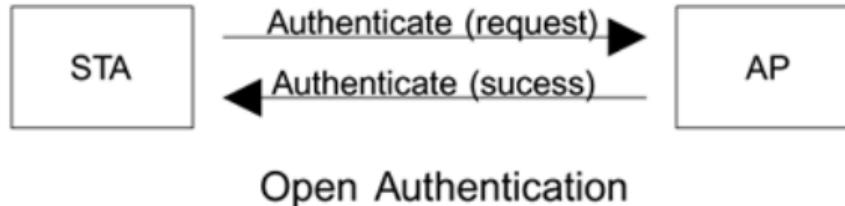
- Para muitos usuários, o WEP foi a única escolha até a adoção de novos métodos de segurança no padrão IEEE 802.11.
- Mesmo com suas fraquezas, o WEP era considerado melhor do que não ter nenhuma proteção.
- O WEP cria uma **barreira mínima**, suficiente para desencorajar atacantes casuais, que preferem procurar redes totalmente abertas.
- **Eficácia do WEP em Ambientes Domésticos:**
  - A maioria dos ataques ao WEP depende da coleta de um número significativo de pacotes transmitidos.
  - Em redes domésticas, onde o tráfego seria a ser pequeno, o WEP **poderia oferecer** um nível razoável de segurança.

# Funcionamento do WEP

- **Fases da segurança no WEP:**
  - Autenticação: o dispositivo prova sua identidade ao ponto de acesso.
  - Criptografia: garante a confidencialidade das mensagens após a autenticação.
- **Propósito da autenticação:**
  - Permitir que cada parte prove que é quem afirma ser.
  - Similar a assinar um cheque: confirma a identidade do remetente.
- **Propósito da criptografia:**
  - Permitir que os dados trafegados não sejam vistos por quem está interceptando (Confidencialidade)
  - Fazer com que quem envie ou receba dados na rede faça-o se possuir a chave.

# Autenticação no WEP

- A autenticação WEP tem o objetivo de provar ao ponto de acesso legítimo que o dispositivo móvel conhece a chave secreta.



# Formato da Mensagem de Autenticação (802.11)

## Campos principais

- **Algorithm Number:** Indica o tipo de autenticação
  - 0 – Open System
  - 1 – Shared Key (WEP)
- **Transaction Sequence:** Número da etapa na sequência de autenticação
  - Mensagem 1, Mensagem 2, (e Mensagem 3 no caso do WEP)
- **Status Code:** Indica sucesso ou falha na autenticação (última mensagem)
- **Challenge Text:** Usado apenas na autenticação por chave compartilhada (WEP)

**Figure 6.2. Authentication Message Format**

Algorithm Num	Transaction Seq.	Status Code	Challenge Text
---------------	------------------	-------------	----------------

## Definição

Processo de autenticação que verifica uma identidade exigindo que uma resposta correta seja fornecida a um **desafio** (*challenge*).

## Como funciona em sistemas computacionais

- O sistema envia um valor imprevisível (desafio).
- O usuário deve calcular e enviar a resposta correta.
- A resposta é validada para confirmar a identidade.

## Autenticação WEP - Mecanismo de Challenge-Response

- Quando o dispositivo solicita autenticação, o ponto de acesso envia um número aleatório chamado **challenge text**.
  - Um número arbitrário de 128 bits, preferencialmente aleatório.
- O dispositivo móvel então criptografa esse número com a chave secreta usando WEP e envia de volta ao ponto de acesso.
- Como o ponto de acesso lembra o número aleatório enviado, ele pode verificar se a resposta foi criptografada com a chave correta.
- O dispositivo deve conhecer a chave para criptografar o valor aleatório corretamente.

# Limitações da Autenticação no 802.11

## Observações importantes:

- Isso não prova ao dispositivo móvel que o ponto de acesso conhece a chave.
- Se um atacante estiver escutando, ele terá um par de *plaintext-ciphertext* para começar ataques.
- Devido a essas vulnerabilidades, a Wi-Fi Alliance abandonou o uso desse mecanismo de autenticação.

## Problemas

- O método de autenticação com WEP é praticamente inútil.
- Na verdade, é pior que inútil, pois fornece ao atacante informações úteis para explorar.

# WEP e o Processo de Autenticação

- Quando o WEP está habilitado, as mensagens de dados são criptografadas.
- Objetivo: impedir que atacantes entendam o conteúdo mesmo escutando o tráfego.
- O padrão IEEE 802.11 original previa duas fases:
  1. Autenticação.
  2. Criptografia dos dados.
- Para decodificar, é necessário conhecer a **chave secreta**.

# RC4 no WEP

- O WEP utiliza a cifra de fluxo **RC4** (Schneier, 1996).
- Funcionamento em alto nível:
  - Entrada: um byte do fluxo de dados por vez.
  - Saída: um byte diferente, aparentemente aleatório.
  - Resultado: a saída parece uma sequência de números aleatórios.
- A descriptografia é o processo inverso.
- O mesmo segredo é usado tanto para criptografar quanto para descriptografar (cifra simétrica).

**Figure 6.3. Stream Cipher**

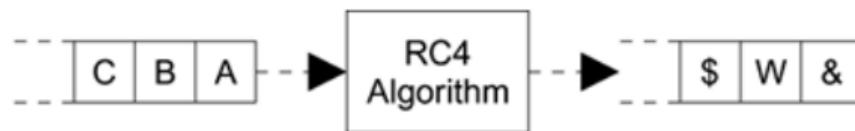


Figure: RC4 como cifra de fluxo no WEP

## RC4 e sua utilização no WEP

- RC4 é simples de implementar, sem operações complexas como multiplicação.
- Possui duas fases principais:
  - **Inicialização:** tabelas internas são construídas a partir da chave.
  - **Encriptação:** os dados passam pelo algoritmo e são cifrados.
- **No WEP, ambas as fases ocorrem a cada pacote transmitido.**
- Essa abordagem garante que a perda de um pacote não comprometa os seguintes.
- Contudo, também introduz vulnerabilidades exploradas em ataques posteriores.
  - **Vocês podem imaginar quais seriam?**

# Problema de usar chave fixa no WEP

- O WEP utiliza o algoritmo RC4.
- Com uma chave fixa: todos os pacotes seriam criptografados com a mesma chave.
- Consequência: textos repetidos geram sempre a mesma saída.
- Isso permite a um atacante identificar padrões, como endereços IP, já que certas partes dos pacotes se repetem.

## Exemplo do Problema

- Texto: "Source Address"
- Resultado cifrado (exemplo): "b%fP\*aF\$!Y"
- Sempre que o mesmo texto for cifrado com a mesma chave, o resultado será idêntico.
- Isso revela informação valiosa para o atacante.

## Solução: Initialization Vector (IV)

- Em vez de usar apenas a chave fixa, o WEP adiciona um número de 24 bits chamado **IV**.
- O IV muda a cada pacote transmitido.
- A chave efetiva passa a ser: **Chave Secreta (104 bits) + IV (24 bits)**.
- Isso garante que pacotes iguais gerem resultados diferentes.

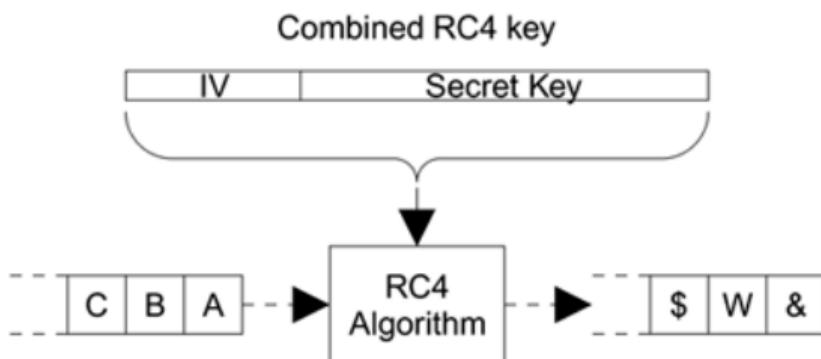
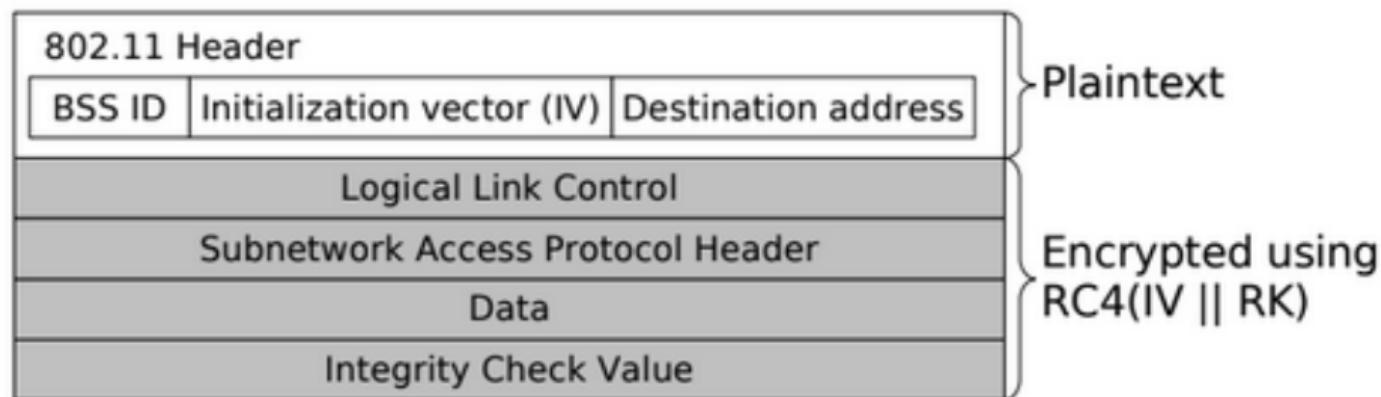


Figure: RC4 com IV e Secret Key

## Limitação do IV

- O IV é transmitido junto com o pacote, logo não é secreto.
- Chamar isso de “128-bit security” é enganoso, pois apenas 104 bits são realmente secretos.
- **Mesmo assim...** o uso do IV **dificulta** a identificação de padrões simples.

**Fig. 1.** A 802.11 frame encrypted using WEP



# Problemas do Vetor de Inicialização (IV) no WEP

- O IV não é secreto: é transmitido abertamente junto com os pacotes.
- **O mesmo IV nunca deveria ser reutilizado com a mesma chave secreta.**
- No WEP, o IV tem apenas **24 bits** ⇒ cerca de 17 milhões de valores possíveis.
- Em um ponto de acesso a 11 Mbps:
  - todos os IVs podem ser usados em poucas horas.
- Reutilização de IVs é inevitável, pois as chaves raramente são trocadas.

## Problemas adicionais:

- Muitos dispositivos reiniciam sempre com o mesmo IV (**Problema de implementação**).
- Sequências “pseudoaleatórias” de IV podem se repetir em diferentes dispositivos.
- Consequência: IVs repetidos facilitam ataques contra o WEP.

# Funcionamento do WEP e Fragmentação

- Aplicações enviam os dados.
- Os dados pode ser dividido em vários fragmentos.
- Cada fragmento é transformado em um **MPDU** (MAC Protocol Data Unit).
- Para cada MPDU:
  - É adicionado um cabeçalho MAC.
  - É incluído um **ICV** no final.
  - Cada fragmento é processado separadamente pelo WEP.
- O processo de criptografia trata o MPDU como um bloco de bytes (10 a 1500 bytes).
- Antes da cifragem, adiciona-se o **Integrity Check Value (ICV)**.

# Integrity Check Value (ICV)

- Objetivo: detectar alterações nos dados durante a transmissão.
- Calculado como um CRC (Cyclic Redundancy Check) de 4 bytes.
- O ICV é adicionado **antes da criptografia** WEP.
- O CRC convencional ainda é adicionado **após a criptografia**.
- Ideia: como o ICV é cifrado, um atacante não poderia alterá-lo.

## Limitações do ICV

- Funciona contra **erros acidentais**, mas não contra ataques ativos.
- Atacantes podem recalcular o CRC após modificar a mensagem.
- Em teoria, o ICV deveria tornar a mensagem *à prova de adulteração*.
- Na prática, há como contornar essa proteção.

## Preparando o frame para transmissão

- Pega os dados e calcula o ICV
- Após o ICV ser anexado, o quadro está pronto para criptografia.
- Seleciona-se um valor de **IV** (24 bits) e o anexa à chave WEP.
- O motor do **RC4** é inicializado.
- Cada byte do bloco (Dados + ICV) é processado → sai um byte criptografado.
- O quadro transmitido contém:
  - IV (3 bytes) + KeyID (1 byte) no início.
  - Dados criptografados + ICV criptografado.
  - Cabeçalho MAC e CRC ao final.
- Um bit no cabeçalho MAC indica que o quadro está protegido por WEP.

## Recebendo o frame WEP

- O receptor identifica o **bit WEP** no cabeçalho.
- Lê o valor do IV (3 bytes) e o KeyID (1 byte).
- Seleciona a chave WEP correta e inicializa o RC4.
- Processa cada byte do fluxo criptografado → obtém os dados originais.
- Como RC4 é simétrico, criptografia = decriptação (dupla aplicação anula o efeito).
- Calcula o ICV e compara com o recebido.
- Se válido, os dados são entregues para processamento superior.

## Falhas do WEP - Autenticação

- Autenticação serve para provar a identidade; idealmente, cada comunicação deve ser autenticada.
- Autenticação mútua é necessária em Wi-Fi:
  - O usuário precisa verificar a rede.
  - A rede precisa verificar o usuário.
- Regras recomendadas para autenticação:
  1. Método robusto que não possa ser falsificado.
  2. Identidade deve persistir em transações subsequentes e não ser transferível.
  3. Autenticação mútua.
  4. Chaves de autenticação devem ser independentes das chaves de criptografia.

# Falhas na autenticação WEP

## Mecanismo: Challenge-response

- 1. O AP envia uma string aleatória de números.
- 2. A estação móvel cifra a string e envia de volta.
- 3. O AP decifra e compara com a string original.

## Problemas:

- **Regra 4 quebrada:** A chave usada na autenticação é a mesma usada para criptografia.
- **Regra 3 quebrada:** O AP não é autenticado; um AP malicioso pode enviar sucesso sem conhecer a chave.
- **Regra 2 quebrada:** Não há token para validar transações futuras.
- **Regra 1 irrelevante:** Devido às falhas anteriores, a robustez da autenticação é comprometida.

# Ataque à autenticação WEP via XOR

**Cenário:** Durante a autenticação, o AP envia um desafio de 128 bytes e a estação móvel responde cifrando com RC4.

**Transação capturada pelo atacante:**

Desafio (P) → Texto claro

$$R = C \oplus P$$

Resposta (C) → Texto cifrado (RC4)

$$\text{Porque } P \oplus R = C \text{ e } C \oplus P = R$$

**Consequência:** O atacante agora conhece o key stream  $R$  correspondente ao IV usado. Ele pode reutilizá-lo em futuras autenticações, conseguindo se autenticar sem conhecer a chave secreta.

**Observação:** Essa falha fornece gratuitamente ao atacante os primeiros 128 bytes do key stream, que são os mais vulneráveis.

# Ataque à Autenticação por Challenge-Response

**Contexto:** O mecanismo de autenticação do WEP usava um desafio (*challenge*) que era criptografado pela estação e enviado de volta ao ponto de acesso.

**Cenário:** Um atacante captura a seguinte troca de mensagens durante uma autenticação WEP:

- Desafio (P) enviado pelo AP: 2A7F9C4E (hexadecimal)
- Resposta (C) enviada pela estação: D3B1AC8B (hexadecimal)

**Pergunta:**

1. Utilizando a operação XOR, calcule o keystream (R) usado pela estação para criptografar o desafio.
2. Explique como um atacante, conhecendo este par (P, C), pode se autenticar fraudulentamente na rede no futuro, sem precisar conhecer a chave secreta WEP.

## Cálculo do keystream (R) usando XOR

$$R = P \oplus C$$

$$R = 2A7F9C4E \oplus D3B1AC8B = F9CE30C5$$

2A7F9C4E = 0010.1010.0111.1111.1001.1100.0100.1110

D3B1AC8B = 1101.0011.1011.0001.1010.1100.1000.1011

---

F9CE30C5 = 1111.1001.1100.1110.0011.0000.1100.0101

Atenção, para não confundir!

Esse R não é a chave secreta da rede WEP, mas sim o keystream (fluxo de chave) que o algoritmo RC4 gerou naquele momento específico, usando a combinação do Vetor de Inicialização (IV) e da chave secreta.

## Exploração pelo atacante

- O atacante agora conhece o **keystream** correspondente ao IV usado nesta autenticação.
- Ele pode reutilizar o keystream com o mesmo IV em futuras autenticações.
- Consequência: consegue se autenticar **sem precisar conhecer a chave secreta WEP**.
- Esse é o ponto inicial para ataques de quebra de WEP.

## Problema da persistência da autenticação no WEP

- A autenticação acontece apenas no início, durante o challenge-response.
- Depois disso, o sistema não emite nenhum tipo de “token” ou prova de identidade que o usuário possa apresentar nas transações subsequentes.
- Então, para todo o resto da comunicação, a rede não revalida a identidade do usuário, e tudo se baseia apenas na chave de criptografia usada para cifrar/decriptar os pacotes.

# Replay Attack e WEP

## Exemplo de ataque:

- Um invasor captura quadros (frames) entre um AP e um dispositivo móvel usando um sniffer.
- Observa mensagens criptografadas e seu tamanho, sem decifrar o conteúdo.
- Quando a usuária legítima se desconecta, o invasor se conecta usando o MAC da vítima.
- O invasor reenvia (replay) uma mensagem previamente capturada.
- O AP aceita a mensagem, passando-a ao servidor, que autentica o invasor sem perceber a fraude.

## Problema:

- WEP não possui proteção contra replay.
- Sequência de número no MAC não é protegida pelo WEP.
- Mensagens antigas podem ser reenviadas sem precisar quebrar a criptografia.
- Consequência: invasor consegue se autenticar indevidamente.

# Falha do WEP na Detecção de Modificação de Mensagens

**Objetivo do WEP:** prevenir alterações nos dados transmitidos usando o *Integrity Check Value (ICV)*.

## Como funciona:

- O ICV é um valor CRC de 32 bits calculado sobre os dados.
- ICV é adicionado ao final da mensagem e toda a sequência é criptografada.
- Teoricamente, alterações no ciphertext seriam detectadas na decriptação.

## Problema:

- O CRC é linear: é possível prever como o ICV muda ao alterar bits da mensagem.
- A operação XOR usada na criptografia permite *bit flipping*: alterar bits no ciphertext altera os mesmos bits na mensagem decifrada.
- Um atacante que conheça os bits certos pode modificar a mensagem e ajustar o ICV correspondente, mantendo a integridade aparente.

**Conclusão:** WEP não fornece proteção efetiva contra modificações no ciphertext.

# Por que o IV é importante no WEP?

## Função do IV (Initialization Vector):

- Adicionado à chave secreta para formar a chave de criptografia usada pelo RC4.
- Garante que duas mensagens idênticas não gerem o mesmo ciphertext.

## E se não houvesse IV?

- RC4 seria inicializado sempre no mesmo estado.
- O fluxo de chave (key stream) seria idêntico para todos os frames.
- Consequência: um atacante que descubra o fluxo de chave de um frame pode descriptografar todos os outros frames.
- Não seria necessário conhecer a chave secreta!

## IV diferente para cada frame

### Por que usar IV diferente em cada frame?

- Adicionando o IV à chave secreta, o RC4 é inicializado em um estado diferente a cada frame.
- Consequência: o fluxo de chave (*key stream*) é diferente para cada criptografia.

### IV constante ou reutilizado é problemático:

- IV constante → mesmo problema do caso de chave fixa.
- IV reutilizado → permite que atacantes detectem padrões e quebrem a criptografia.
- Reuso inevitável se o número de IVs possíveis é limitado (24 bits no WEP).
- Seleção aleatória de IVs aumenta o risco de repetição cedo (paradoxo do aniversário).

# Paradoxo do Aniversário e IVs no WEP

## Paradoxo do Aniversário:

- Probabilidade de pelo menos duas pessoas compartilharem o mesmo aniversário.
- Surpreendentemente, com apenas 23 pessoas, a probabilidade > 50%.
- Explicação: existem  $23 \times 22/2 = 253$  pares possíveis.

## Aplicação em criptografia:

- Ataques de colisão (*birthday attack*) em funções hash.
- Similarmente, reuso de IVs em WEP aumenta a chance de colisões no fluxo de chave.

**Lição para WEP:** O número limitado de IVs (24 bits) faz com que colisões aconteçam cedo, tornando o sistema vulnerável.

# Reuso de IVs no WEP

**Problema:** IVs de 24 bits são limitados e podem se repetir rapidamente.

**Solução ideal:** incrementar o IV a cada quadro enviado para maximizar o tempo até uma repetição.

**Exemplo prático:**

- IV de 24 bits → 16.777.216 valores possíveis.
- IEEE 802.11b transmite 500 quadros/s.
- Espaço de IVs esgotado em cerca de 7 horas.

**Problemas reais:**

- Vários dispositivos usando o mesmo IV/ chave → colisões mais cedo.
- Erros de implementação: alguns fabricantes reiniciam o IV em 0 após reinício.
- Resultado: repetição de IV é comum e vulnerabilidade explorável.

# Aula 8 - Números Pseudoaleatórios

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação  
Universidade Federal do Rio de Janeiro  
[gabrielaquino@ic.ufrj.br](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 2025

# Geração de Números Aleatórios em Criptografia

**Função importante:** geração de números pseudoaleatórios criptograficamente fortes.

- Os **Geradores de Números Pseudoaleatórios (PRNGs)** são essenciais para aplicações de criptografia e segurança.
- Eles produzem sequências de números que parecem aleatórias, mas são geradas a partir de um estado inicial (*seed*).

**Importância dos números aleatórios:**

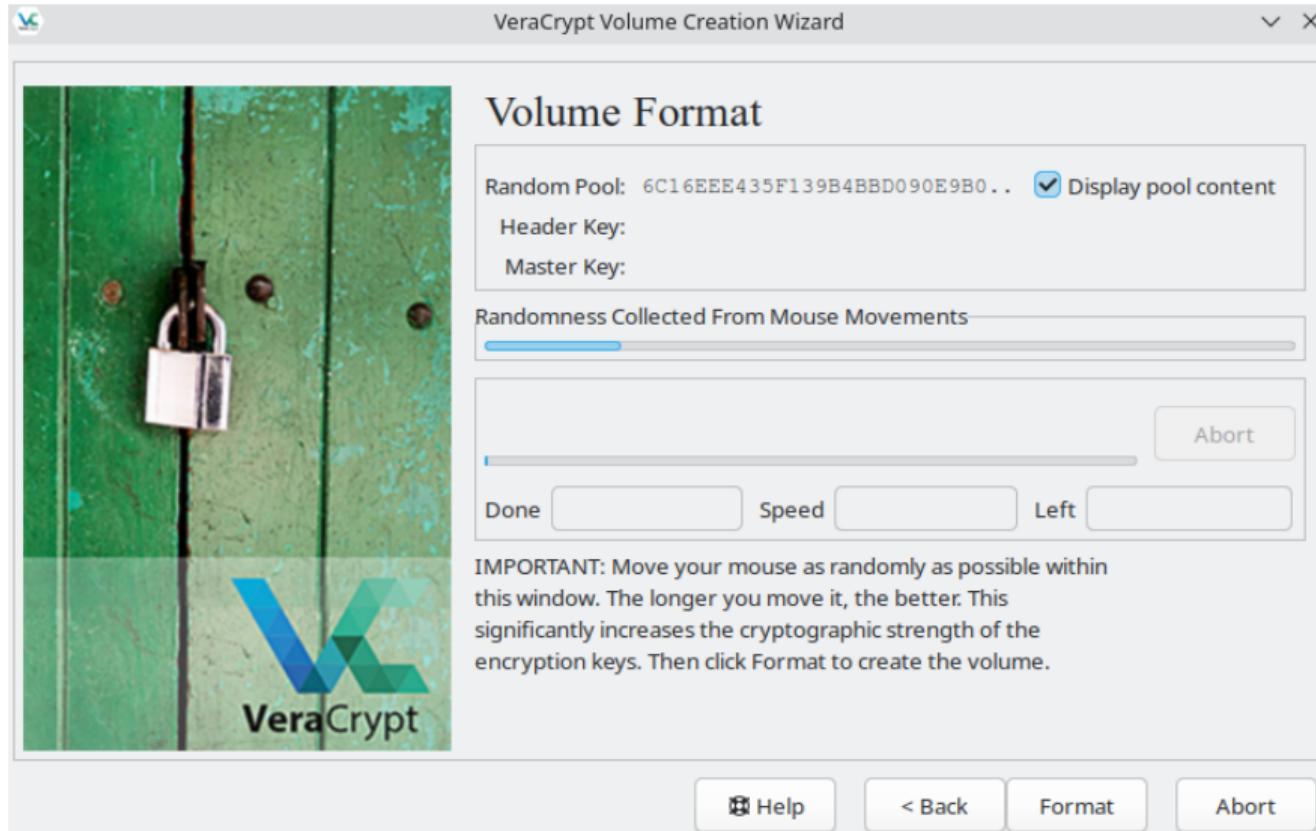
- Garantir **confidencialidade** em criptografia de chave simétrica.
- Produzir **nonces**, IVs e chaves seguras.
- Evitar padrões previsíveis que possam ser explorados por atacantes.
- Fundamentais para diversas aplicações de segurança em redes.

# Uso de Números Aleatórios em Segurança de Redes

**Função:** garantir a segurança em diversos algoritmos criptográficos por meio de números binários aleatórios.

- **Distribuição de chaves e autenticação mútua:**
  - Protocolos cooperativos trocam mensagens para distribuir chaves e autenticar participantes.
  - **Nonces** são usados durante o handshaking para evitar ataques de replay.
  - Números aleatórios nos nonces impedem que atacantes adivinhem ou reutilizem transações antigas.
- **Geração de chave de sessão:**
  - Chaves temporárias para criptografia simétrica, válidas apenas durante uma sessão.
  - Aumenta a segurança limitando a exposição da chave.
- **Geração de chaves para RSA:** Números aleatórios são essenciais para gerar chaves de criptografia assimétrica seguras.
- **Fluxos de bits para cifragem de fluxo:** Algoritmos de cifra de fluxo dependem de números aleatórios para criar sequências criptograficamente seguras.

# Exemplo: VeraCrypt tem necessidade de Números Aleatórios



# Critérios para Números Aleatórios

Tradicionalmente, a preocupação na geração de uma sequência de números supostamente aleatórios é garantir que a sequência seja estatisticamente aleatória.

Dois critérios principais são usados para validar a aleatoriedade:

- **Distribuição uniforme:** A frequência de ocorrência de uns e zeros deve ser aproximadamente a mesma, garantindo que não haja viés.
- **Independência:** Nenhum valor na sequência pode ser deduzido a partir dos outros; cada elemento é estatisticamente independente dos demais.

# Testando Independência de Sequências Aleatórias

Embora existam testes bem definidos para verificar se uma sequência de bits segue uma distribuição específica, como a **uniforme**, não há um teste único para **provar independência**.

Estratégia geral:

- Aplicar diversos testes estatísticos que verificam propriedades relacionadas à independência.
- Se nenhum teste indicar dependência, podemos ter um **alto nível de confiança** de que a sequência é independente.
- A confiança aumenta com o número e a variedade de testes aplicados.

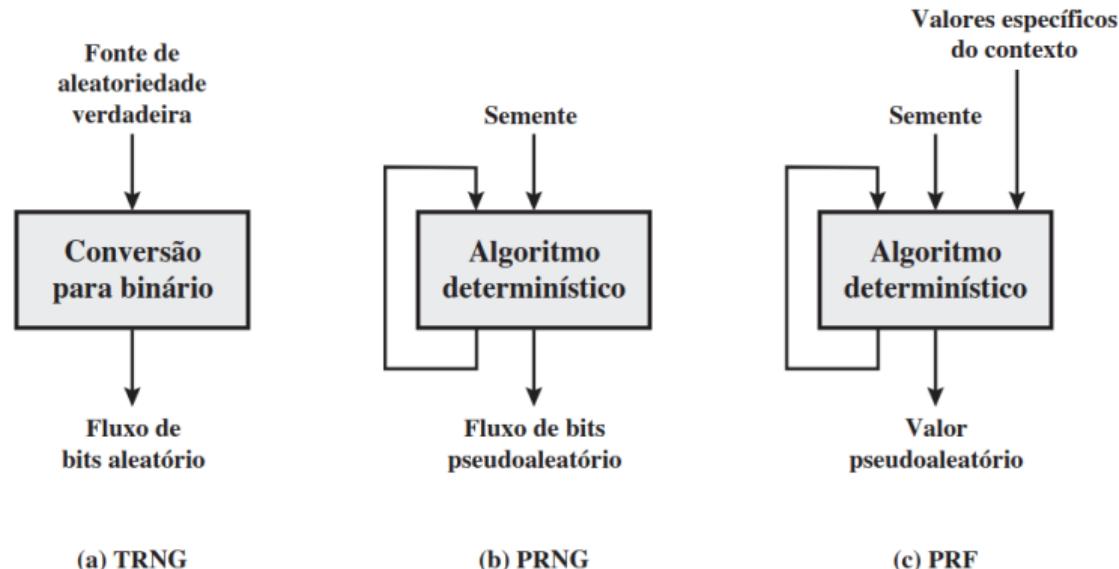
# Imprevisibilidade em Números Aleatórios

Em aplicações como autenticação recíproca, geração de chaves de sessão e cifras de fluxo, o requisito principal não é a aleatoriedade estatística, mas a **imprevisibilidade** dos números sucessivos.

- Números verdadeiramente aleatórios: cada elemento é independente e, portanto, imprevisível.
- Limitações dos números verdadeiramente aleatórios: ineficiência e dificuldade de geração.
- Solução comum: algoritmos geradores de números pseudoaleatórios (**PRNGs**) que aparentam aleatoriedade.
- Importante: garantir que um atacante não consiga prever elementos futuros da sequência a partir dos anteriores.

# Geradores de números aleatórios e pseudoaleatórios

**Figura 7.1** Geradores de números aleatórios e pseudoaleatórios.



TRNG = gerador de número aleatório verdadeiro (true random number generator)

PRNG = gerador de número pseudoaleatório (pseudorandom number generator)

PRF = função pseudoaleatória (pseudorandom function)

**Geração de números aleatórios em criptografia:** normalmente realizada por algoritmos determinísticos.

- **Determinismo:** Algoritmos produzem sequências que não são estatisticamente aleatórias.
- **Números pseudoaleatórios:** Se o algoritmo for bom, a sequência resultante passa em muitos testes de aleatoriedade.
- **Prática aceita:** Apesar de serem determinísticos, PRNGs funcionam “tão bem quanto” números aleatórios em aplicações criptográficas.
- **Analogias estatísticas:** Semelhante a amostras de uma população — resultados aproximam os da população inteira.

## Geradores de números aleatórios verdadeiros (TRNG):

- Usam uma fonte efetivamente aleatória, chamada **fonte de entropia**.
- Fontes de entropia podem incluir:
  - Padrões de temporização dos toques de tecla.
  - Movimentos do mouse.
  - Atividade elétrica no disco.
  - Valores instantâneos do clock do sistema.
- A fonte de entropia serve como entrada para um algoritmo que gera saída binária aleatória.
- Pode envolver conversão de sinais analógicos e processamento adicional para eliminar tendências.

## TRNG – True Random Number Generator

A fonte de entropia é retirada do ambiente físico do computador e pode incluir:

- Padrões de temporização dos toques de tecla
- Atividade elétrica no disco
- Movimentos do mouse
- Valores instantâneos do clock do sistema

A saída é binária e aleatória, podendo envolver conversão de fonte analógica e processamento para reduzir tendências.

## PRNG – Pseudo-Random Number Generator

Toma como entrada um valor fixo, chamado semente, e produz uma sequência de bits determinística.

- A semente frequentemente é gerada por um TRNG.
- Bits adicionais podem ser produzidos realimentando parte da saída como entrada.
- O fluxo de saída é totalmente determinado pela semente e pelo algoritmo, permitindo reprodução completa se ambos forem conhecidos.

## Gerador de Número Pseudoaleatório (PRNG)

Um algoritmo usado para produzir uma sequência de bits tão grande quanto necessário.

- Aplicações típicas: entrada para cifras de fluxo simétricas.
- Pode gerar longas sequências determinísticas a partir de uma semente inicial.
- Consulte a Figura 3.1a para ilustração do fluxo.

## Função Pseudoaleatória (PRF)

Produz uma cadeia de bits pseudoaleatória com comprimento fixo.

- Aplicações típicas: geração de chaves de encriptação simétrica e nonces.
- Entrada: semente + valores específicos do contexto (ex.: ID de usuário ou ID de aplicação).

# Requisitos de um PRNG/PRF em Criptografia

- O requisito básico: um adversário que não conhece a semente não deve ser capaz de determinar a sequência pseudoaleatória.
- Exemplo em cifras de fluxo: conhecer o fluxo pseudoaleatório permite recuperar o texto claro a partir do texto cifrado.
- Exemplo em PRFs: uma semente de 128 bits e valores de contexto geram uma chave secreta de 128 bits para encriptação simétrica. Saída não suficientemente aleatória facilita ataques por força bruta.
- Implicações: a segurança da saída depende de aleatoriedade, imprevisibilidade e das características da semente.

- O fluxo de bits gerado por um PRNG deve **parecer aleatório**, mesmo sendo determinístico.
- Nenhum teste isolado pode provar aleatoriedade; aplica-se uma **sequência de testes**.
- Segundo o NIST SP 800-22, três características devem ser avaliadas:
  - **Uniformidade**: zeros e uns ocorrem com probabilidade  $1/2$ ; o número esperado de zeros (ou uns) é  $n/2$ , onde  $n$  é o tamanho da sequência.
  - **Escalabilidade**: subsequências extraídas aleatoriamente também devem passar nos testes de aleatoriedade.
  - **Consistência**: o comportamento do gerador deve ser coerente entre diferentes sementes.
- [Veja os testes do NIST](#)
- [Software de testes do NIST](#)
- Não é adequado testar um PRNG usando apenas uma única semente ou um TRNG usando apenas uma saída física.

# Testes de Aleatoriedade do NIST SP 800-22

- O SP 800-22 lista 15 testes separados para avaliar a aleatoriedade de um PRNG.
- Aqui estão três exemplos e seus propósitos:
  - **Teste de frequência:** verifica se o número de uns e zeros na sequência é aproximadamente o esperado para uma sequência verdadeiramente aleatória.
  - **Teste de rodadas:** avalia o número total de rodadas (sequências consecutivas de bits idênticos limitadas por bits opostos) para ver se correspondem ao esperado.
  - **Teste da estatística universal de Maurer:** mede a distância entre padrões correspondentes; identifica se a sequência é significativamente comprimível e, portanto, não aleatória.
- Esses testes fornecem uma visão geral do comportamento estatístico das sequências pseudoaleatórias.

# Imprevisibilidade de Números Pseudoaleatórios

Um fluxo de números pseudoaleatórios deve exibir duas formas de imprevisibilidade:

- **Imprevisibilidade direta:** se a semente é desconhecida, o próximo bit da sequência deve ser imprevisível, mesmo conhecendo todos os bits anteriores.
- **Imprevisibilidade inversa:** não é viável determinar a semente a partir de quaisquer valores gerados. Nenhuma correlação entre a semente e os bits gerados deve ser aparente.
- O mesmo conjunto de testes de aleatoriedade também verifica a imprevisibilidade.
- Se a sequência parece aleatória, não é possível prever bits futuros nem deduzir a semente a partir da sequência conhecida.

# Requisitos da Semente para PRNGs Criptográficos

Para aplicações criptográficas, a semente de um PRNG deve ser segura:

- **Imprevisível:** se o adversário puder deduzir a semente, poderá reproduzir toda a saída do PRNG.
- Normalmente gerada por um **TRNG**, garantindo que a semente seja aleatória ou pseudoaleatória.
- Em cifras de fluxo, o TRNG não é prático para gerar o fluxo de chave completo; o PRNG permite transmitir apenas a chave curta com segurança.
- Para funções pseudoaleatórias (PRFs), o TRNG fornece a semente, e a PRF gera os bits de saída para eliminar possíveis vieses do TRNG.
- O TRNG pode ter limitações de velocidade; o PRNG permite gerar números suficientes para a aplicação.

# PRNG e a seed

**Figura 7.2** Geração de entrada de semente ao PRNG.



# Projeto de Algoritmos de PRNG Criptográficos

PRNGs criptográficos podem ser classificados em duas categorias principais:

- **Algoritmos de propósito especial:** projetados especificamente para gerar fluxos de bits pseudoaleatórios. Alguns são usados para várias aplicações de PRNG; outros são dedicados a cifras de fluxo, como o **RC4**.
- **Algoritmos baseados em algoritmos criptográficos existentes:** utilizam propriedades de algoritmos criptográficos para produzir aleatoriedade.

Três técnicas gerais para gerar PRNGs criptograficamente fortes:

- **Cifras de bloco simétricas**
- **Cifras assimétricas**
- **Funções de hash e códigos de autenticação de mensagem**

Essas técnicas podem ser combinadas e reutilizadas de forma prática, aproveitando algoritmos já existentes em sistemas de encriptação e autenticação.

**Geradores de números aleatórios verdadeiros (TRNGs) utilizam fontes não determinísticas:**

- Medem processos naturais imprevisíveis, como:
  - Detectores de pulso de radiação ionizante.
  - Tubos de descarga de gás.
  - Capacitores com escape.
- Exemplos comerciais e de pesquisa:
  - Chip da Intel: captura ruído térmico, amplificando voltagem de resistores não condutíveis.
  - **LavaRnd**: projeto de código aberto que gera números verdadeiramente aleatórios usando câmeras baratas e CCD saturado como fonte caótica. [Link](#)
- O software processa as amostras e produz números aleatórios em vários formatos.

**Possíveis fontes de aleatoriedade para gerar sequências verdadeiramente aleatórias:**

- **Entrada de som/vídeo:**
  - Microfone sem entrada ou câmera tampada produz ruído térmico.
  - Digitalização fornece bits aleatórios de qualidade razoável.
- **Unidades de disco:**
  - Pequenas flutuações aleatórias na rotação devido à turbulência do ar.
  - Medições de tempo de busca geram dados aleatórios, embora correlacionados.
  - Processamento adequado pode extrair excelentes bits aleatórios (ex.: 100 bits/min em discos抗igos).
- Serviço on-line [random.org](http://random.org) que pode oferecer sequências aleatórias pela Internet.
- Exemplo de TCC no tema de geração de números aleatórios: [Link](#)

# Geradores de números aleatórios e pseudoaleatórios

**Tabela 7.5** Comparação entre PRNGs e TRNGs.

	<b>Geradores de números pseudoaleatórios (PRNGs)</b>	<b>Geradores de números aleatórios veradeiros (TRNGs)</b>
<b>Eficiência</b>	Muito eficientes	Geralmente ineficientes
<b>Determinismo</b>	Determinísticos	Não determinísticos
<b>Periodicidade</b>	Periódicos	Aperiódicos

## Características principais dos PRNGs:

- **Eficiência:** podem gerar muitos números rapidamente, adequado para aplicações que exigem grandes quantidades de números pseudoaleatórios.
- **Determinismo:** a mesma sequência pode ser reproduzida se a condição inicial (semente) for conhecida.
- **Periodicidade:** as sequências eventualmente se repetem, mas PRNGs modernos possuem períodos tão longos que são irrelevantes para a maioria das aplicações.

## Características principais dos TRNGs:

- **Eficiência:** geralmente mais lentos que PRNGs, podendo ser um gargalo em aplicações que exigem muitos números aleatórios por segundo.
- **Não determinísticos:** a mesma sequência não pode ser reproduzida de forma garantida.
- **Sem periodicidade:** não possuem ciclo repetitivo, diferentemente dos PRNGs.
- **Aplicações críticas:** úteis em contextos onde imprevisibilidade máxima é necessária, como criptografia bancária ou nacional.

# Propensão em TRNGs e Algoritmos Antipropensão

**Propensão:** um TRNG pode gerar uma saída tendenciosa, com mais uns ou zeros.

**Soluções:**

- Aplicação de **algoritmos antipropensão** para reduzir ou eliminar a tendência.
- Uso de **funções de hash** (MD5, SHA-1) para misturar blocos de bits:
  - Blocos de  $m \geq n$  bits de entrada processados para gerar  $n$  bits de saída.
  - Mistura de entradas de diferentes fontes de hardware.
- Sistemas operacionais oferecem mecanismos embutidos:
  - Linux: combina atividade do mouse/teclado, E/S de disco e interrupções. A saída aleatória passa por SHA-1 antes de ser fornecida.
  - [Leia](#)

# Gerador de Números Aleatórios Digitais da Intel (DRNG)

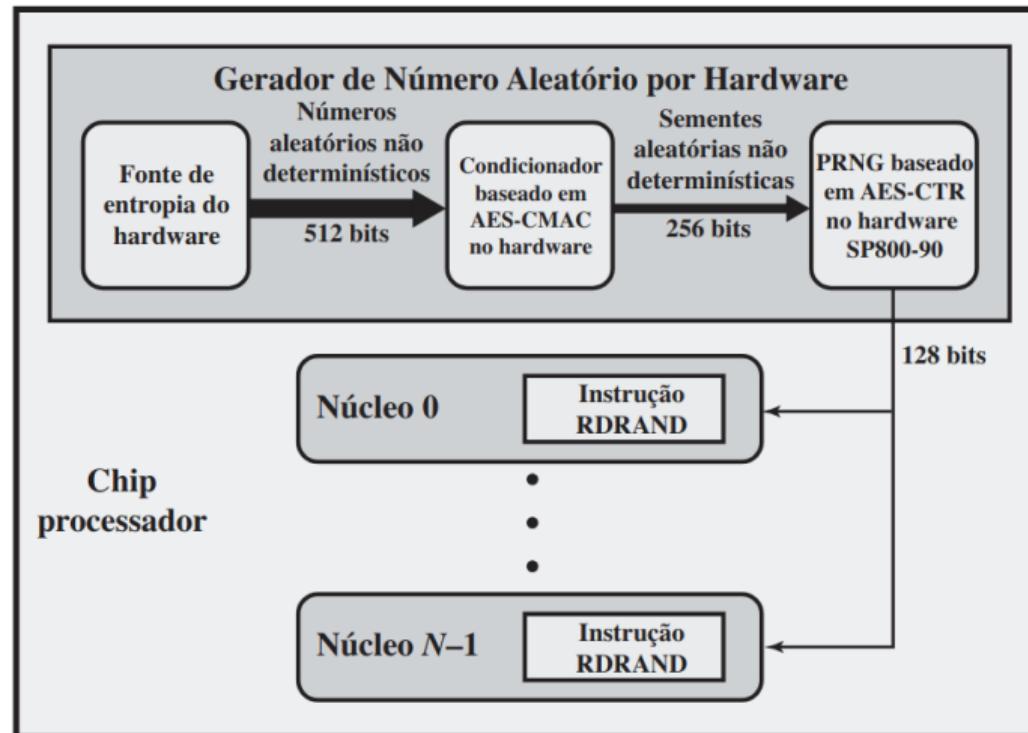
**Contexto:** TRNGs tradicionais eram usados apenas para gerar pequenas quantidades de bits aleatórios devido à baixa taxa de produção.

**Intel DRNG:** primeiro TRNG comercial com taxa de produção comparável a PRNGs, disponível em chips multicore desde 2012.

## Destaques:

- **Implementação totalmente em hardware:** maior segurança e velocidade comparada a soluções baseadas em software.
- **Integração no chip multicore:** elimina atrasos de E/S encontrados em outros TRNGs.

**Figura 7.9** Chip processador da Intel com gerador de números aleatórios.



O DRNG é uma arquitetura de **três estágios**:

1. **Fonte de Entropia**: Gera bits aleatórios a partir do ruído térmico.
2. **Condicionador**: Remove qualquer viés ou correlação residual.
3. **PRNG**: Expande a aleatoriedade para maior throughput.

## Estágio 1: Fonte de Entropia (O Núcleo Aleatório)

- **Núcleo:** Dois inversores (portas NOT).
- Possui dois estados estáveis lógicos.
- Pulses de *clock* forçam ambos para um **estado metaestável** indeterminado.
- O **ruído térmico** aleatório dentro dos transistores decide para qual estado estável o circuito irá decair.
- Este decaimento é fundamentalmente **imprevisível**.

### Desempenho

Gera bits a uma taxa de **4 Gbps** (Gigabits por segundo). A saída é colhida em blocos de **512 bits**.

## Estágio 2: Condicionamento com CMAC

- **Problema:** A saída do estágio 1 pode ter **viés** ou ter **correlações** sutis.
- **Solução:** Um condicionador criptográfico.
- **Função:** CBC-MAC (CMAC), conforme padrão NIST SP 800-38B.

### Como funciona?

- Os 512 bits do Estágio 1 são criptografados em modo **CBC** usando o algoritmo AES.
- Apenas o **último bloco cifrado** (o MAC) é tomado como saída.
- Esta etapa distila entropia, produzindo **256 bits** verdadeiramente aleatórios e não enviesados por bloco.

## Estágio 3: Geração em Alta Velocidade (PRNG)

- **Problema:** Mesmo a 4 Gbps, a entropia pura não é suficientemente rápida para todas as aplicações modernas.
- **Solução:** Usar os bits aleatórios do Estágio 2 como **semente** para um PRNG.

### Algoritmo: CTR\_DRBG

#### Counter Mode Deterministic Random Bit Generator

- **Selete:** Os 256 bits do Estágio 2.
- **Funcionamento:** Criptografa um contador incremental usando AES.
- **Saída:** Gera números pseudoaleatórios de **128 bits**.
- **Segurança:** Sem a semente, é computacionalmente inviável prever a saída.

### Vantagem

A partir de uma única semente de 256 bits, o PRNG pode gerar **muitos** blocos de 128 bits, ultrapassando a taxa da fonte de entropia (**>3 Gbps**). Um limite de **511 amostras** é definido por semente para garantir a segurança antes de uma nova ressementeação.

# Interface com o Software: A Instrução RDRAND

- A saída final do DRNG (seja do Estágio 2 ou 3) é disponibilizada para todos os núcleos do processador.
- Acesso via uma instrução de assembly dedicada: **RDRAND**.

## Instrução RDRAND

RDRAND reg

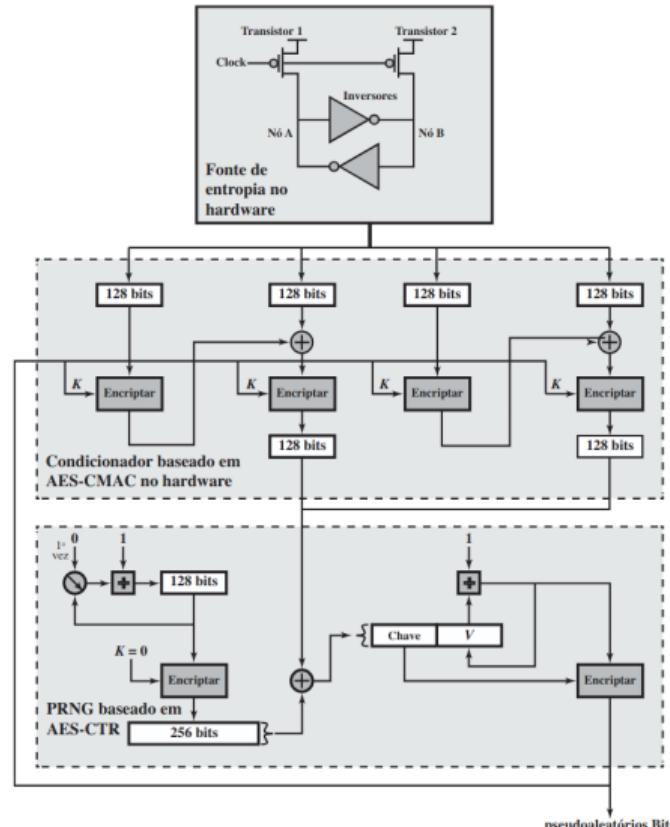
onde reg pode ser um registrador de 16, 32 ou 64 bits (AX, EAX, RAX).

- A instrução busca um valor aleatório do tamanho solicitado.
- Define a flag de *carry* ( $CF=1$ ) em caso de sucesso.
- Software deve verificar o *carry* para garantir que obteve um valor aleatório válido.

[Leitura interessante - Link da Intel](#)

# Intel DRNG: Estrutura lógica

Figura 7.10 Estrutura lógica do DRNG da Intel.



# Código executando o rdrand

```
section .text
    global _start

_start:
    rdrand rcx          ; Bota valor aleatorio no registrador RCX - 64 bits
    jnc .exit            ; Vai pular se NÃO tiver o carry flag (CF = 0)

    ; Escreve os 64 bits diretamente na saída padrao (terminal)
    push rcx             ; coloca os 64 bits na stack
    mov rax, 1            ; Fala qual é a operacão - sys_write
    mov rdi, 1            ; Fala aonde que vai escrever - stdout
    mov rsi, rsp           ; Fala onde que é pra ler - aponta leitura para os dados na stack (rsp é o stack pointer)
    mov rdx, 8             ; Fala a quantidade pra ler - 8 bytes (64 bits)
    syscall

    add rsp, 8            ; Joga o stack pointer 8 bytes para cima - ou seja, limpa a stack

.exit:
    mov rax, 60            ; Fala qual que é a operacão - no caso sys_exit
    xor rdi, rdi           ; Bota zero no rdi - exit(0)
    syscall
```

## Compilando o código com rdrand

```
$nasm -f elf64 rdrand_bin.asm -o rdrand_bin.o  
$ld rdrand_bin.o -o rdrand_bin  
$ ./rdrand_bin | hexdump -C
```

# Aula 9 - Cifras de bloco

Prof. Gabriel Rodrigues Caldas de Aquino

Instituto de Computação  
Universidade Federal do Rio de Janeiro  
[gabrielaquino@ic.ufrj.br](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 2025

# Cifras de Fluxo vs Cifras de Bloco

## Cifras de Fluxo:

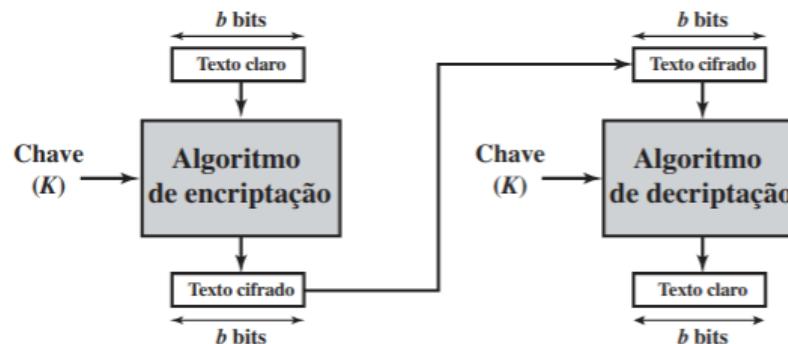
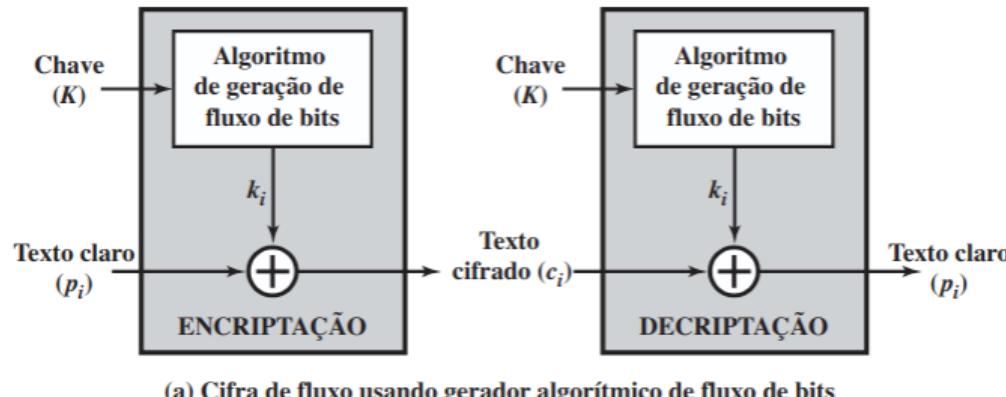
- Encriptam dados um **bit** ou **byte** por vez.
- Exemplos clássicos: Vigenère autochaveada e Vernam.
- Ideal: One-time pad (inquebrável se o fluxo de chaves for verdadeiramente aleatório).
- Limitação prática: o fluxo de chaves precisa ser compartilhado previamente via canal seguro.
- Solução prática: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



# 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  $M = m_1, m_2, m_3, \dots$ :

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

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:**

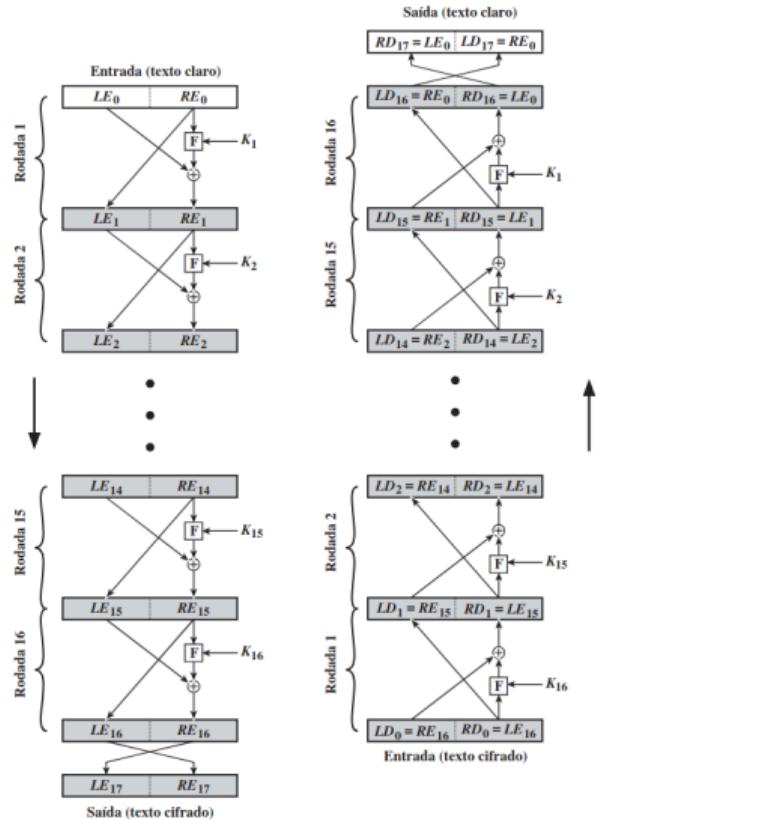
- Complica o relacionamento estatístico entre o texto claro e o texto cifrado.
- Faz com que cada bit do texto claro influencie muitos bits do texto cifrado.
- Frustra tentativas de deduzir padrões no texto claro.

## **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.

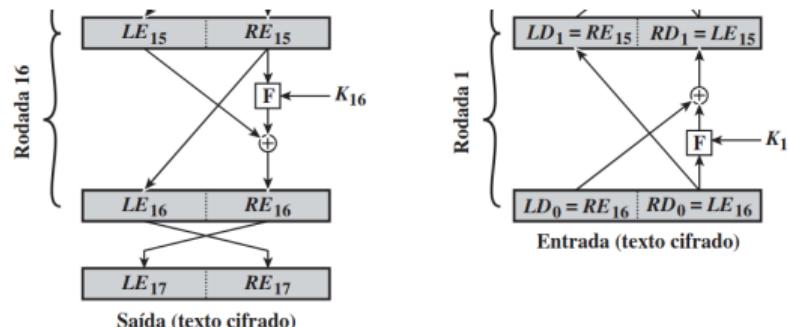
Ou seja:

- 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

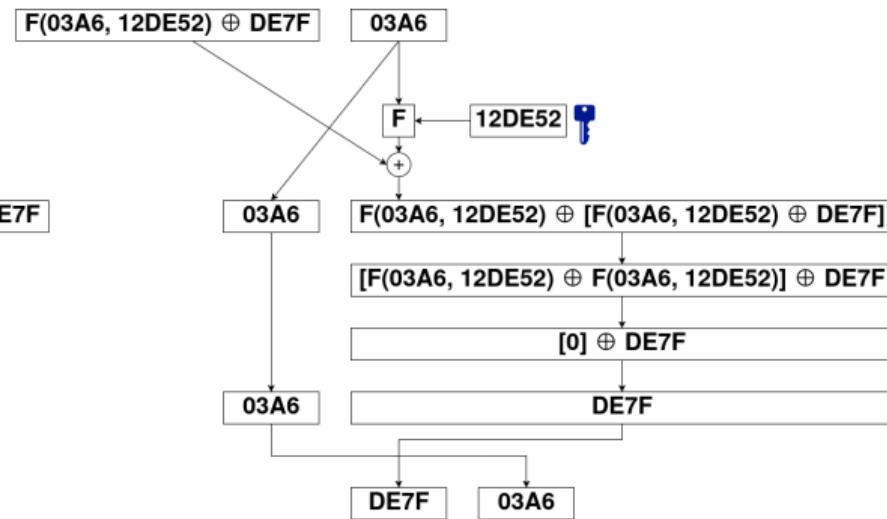
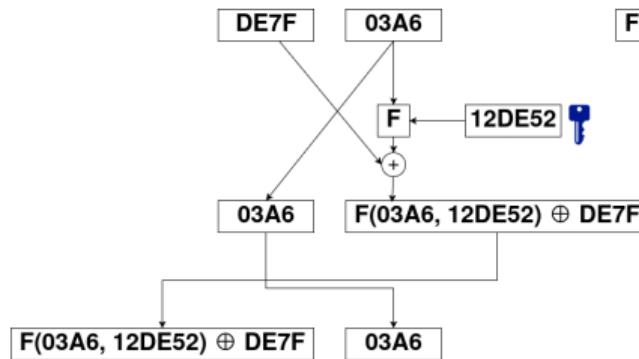


E sabendo que a operação lógica ou-exclusivo tem as seguintes propriedades:

$$\begin{aligned}[A \oplus B] \oplus C &= A \oplus [B \oplus C] \\ D \oplus D &= 0 \\ E \oplus 0 &= E\end{aligned}$$

# Validação do Algoritmo de Feistel com Ordem de Chaves Invertida

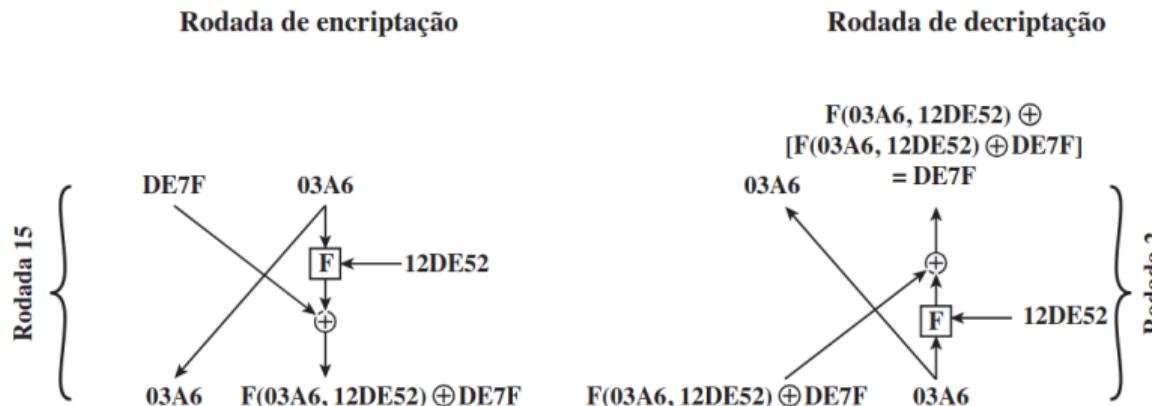
Temos:



# 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:

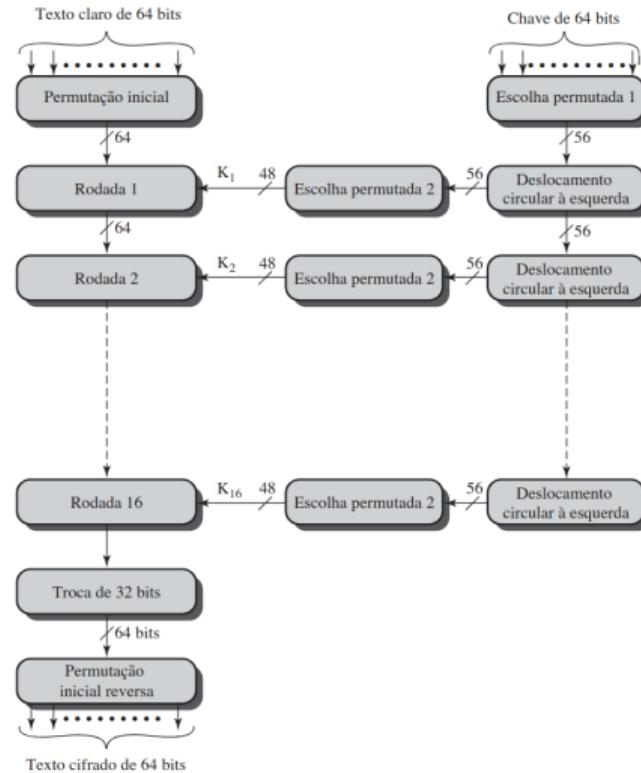
- Adotado em 1977 pelo **NIST**.
- Também conhecido como **Data Encryption Algorithm (DEA)**.
- Encripta dados em blocos de 64 bits usando uma chave de 56 bits.
- O mesmo algoritmo e chave são usados para encriptação e decriptação.

## 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).

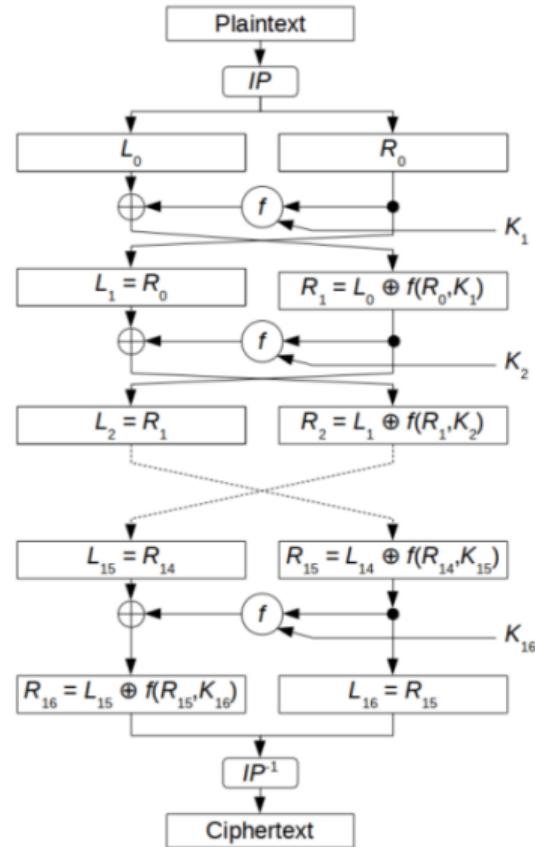
## Processamento do texto claro:

- **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.

## 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 16<sup>a</sup> 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**.

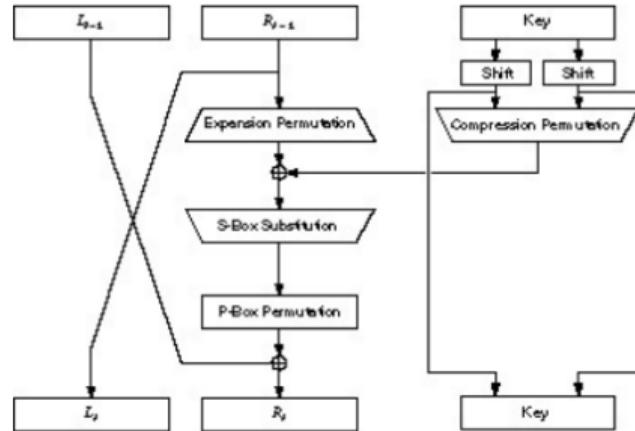
## Equações de uma Rodada no DES

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

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

- $L_i, R_i$ : 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	123a2d0d04262alc	18b3fa41	9616fe23
7	021f120b1c130611	9616fe23	67117cf2
8	1c10372a2832002b	67117cf2	c11bf09

9	04292a380c341f03	c11bf09	887fb06c
10	2703212607280403	887fb06c	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		$\delta$
	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

Rodada		$\delta$
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 <sup>-1</sup>	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		$\delta$
	02468aceeca86420 02468aceeca86420	0
1	3cf03c0fbd22845 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

Rodada		$\delta$
9	c11bfc09887fbc6c 548f1de471f64dfd	34
10	887fbcc6c600f7e8b 71f64dfd4279876c	36
11	600f7e8bf596506e 4279876c399fdc0d	32
12	f596506e738538b8 399fdc0d6d208dbb	28
13	738538b8c6a62c4e 6d208dbbb9bdeeeaa	33
14	c6a62c4e56b0bd75 b9bdeeeaad2c3a56f	30
15	56b0bd7575e8fd8f d2c3a56f2765c1fb	33
16	75e8fd8f25896490 2765c1fb01263dc4	30
IP <sup>-1</sup>	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 microsegundo 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/ $\mu$ s.
  - 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:  $5 \times 10^8$  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	$2^{56} \approx 7,2 \times 10^{16}$	$2^{55} \text{ ns} = 1,125 \text{ ano}$	1 hora
128	AES	$2^{128} \approx 3,4 \times 10^{38}$	$2^{127} \text{ ns} = 5,3 \times 10^{21} \text{ anos}$	$5,3 \times 10^{17} \text{ anos}$
168	Triple DES	$2^{168} \approx 3,7 \times 10^{50}$	$2^{167} \text{ ns} = 5,8 \times 10^{33} \text{ anos}$	$5,8 \times 10^{29} \text{ anos}$
192	AES	$2^{192} \approx 6,3 \times 10^{57}$	$2^{191} \text{ ns} = 9,8 \times 10^{40} \text{ anos}$	$9,8 \times 10^{36} \text{ anos}$
256	AES	$2^{256} \approx 1,2 \times 10^{77}$	$2^{255} \text{ ns} = 1,8 \times 10^{60} \text{ anos}$	$1,8 \times 10^{56} \text{ ano}$
26 caracteres (permutação)	Monoalfabético	$2! = 4 \times 10^{26}$	$2 \times 10^{26} \text{ ns} = 6,3 \times 10^9 \text{ anos}$	$6,3 \times 10^6 \text{ anos}$

## Número de Rodadas no DES

- Maior número de rodadas → 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.

## 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.

## 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)**:

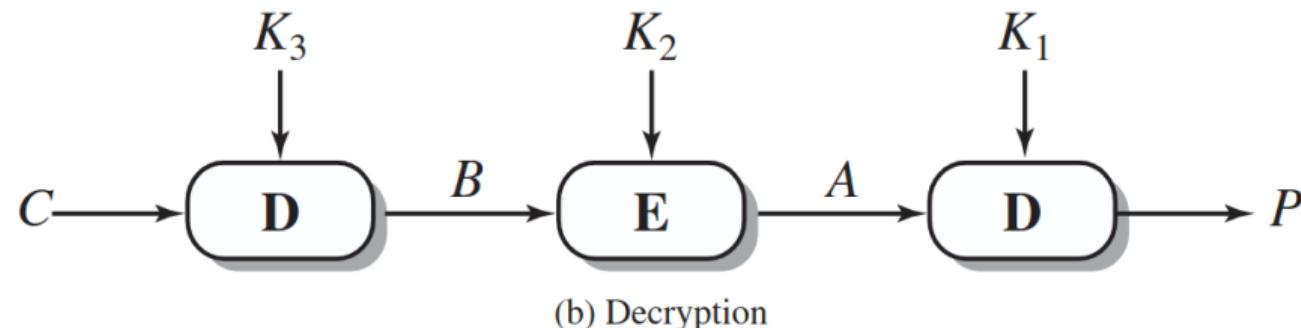
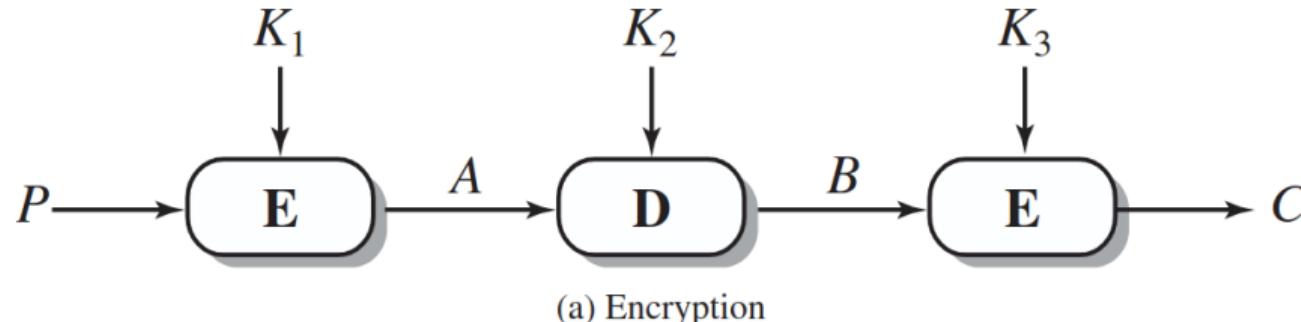
$$C = E(K_3, D(K_2, E(K_1, P)))$$

- $C$  = 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**

	<b>Key length</b>		<b>Block size</b>		<b>Number of rounds</b>
	<i>Nk</i> (in bits)		<i>Nb</i> (in bits)		<i>Nr</i>
<b>AES-128</b>	4	128	4	128	10
<b>AES-192</b>	6	192	4	128	12
<b>AES-256</b>	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 ( $P_1, P_2, \dots, P_N$ ) 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

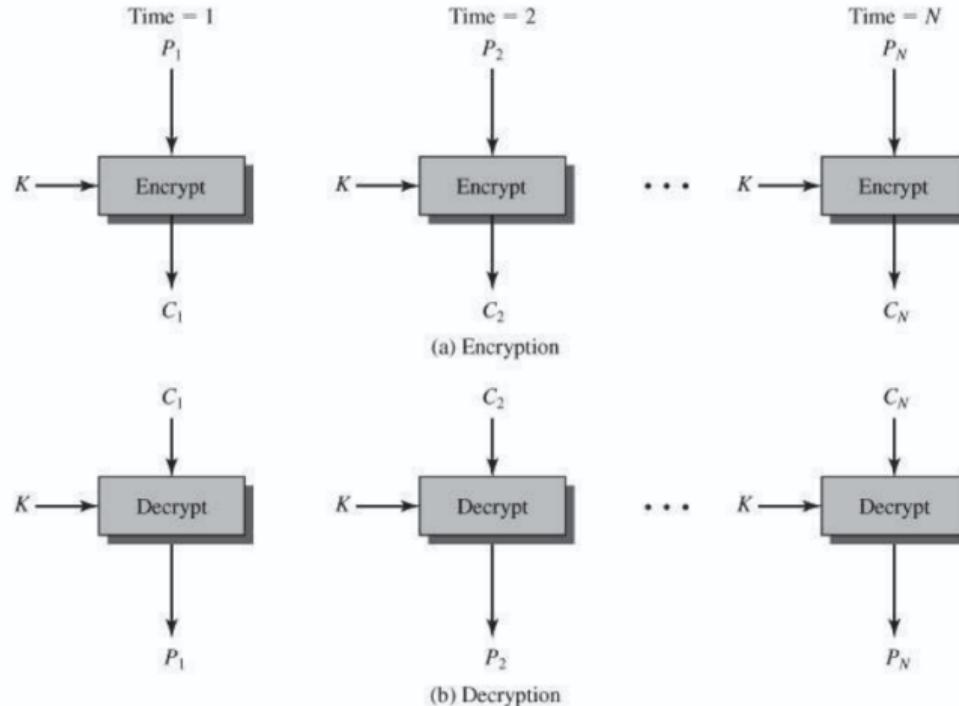


Figure 1: Scheme of the ECB mode of operation [2]

## 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

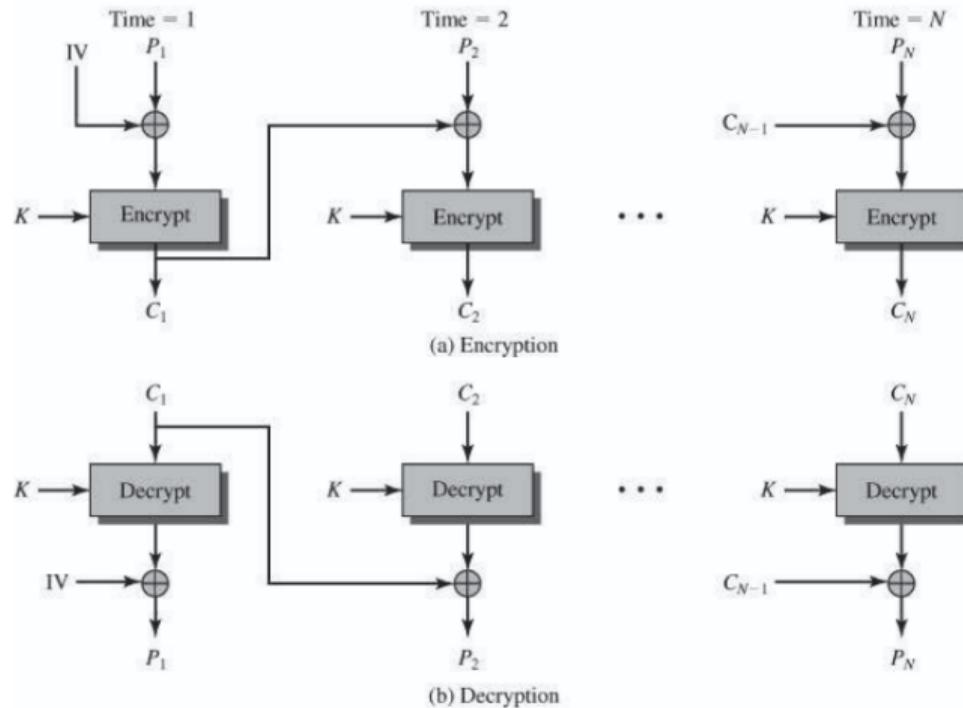
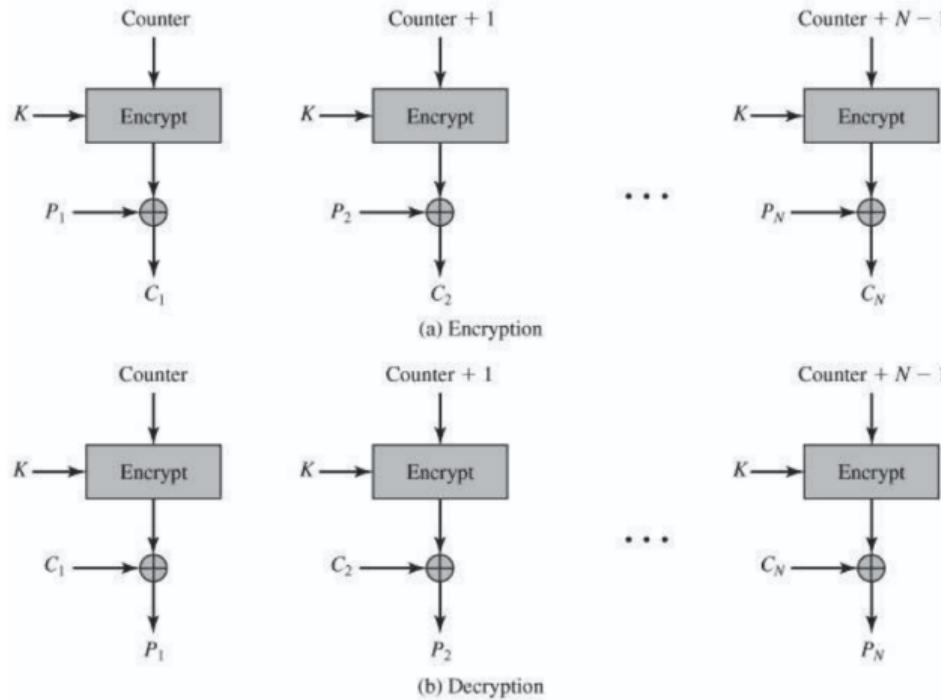


Figure 2: Scheme of the CBC mode of operation [2]

## 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



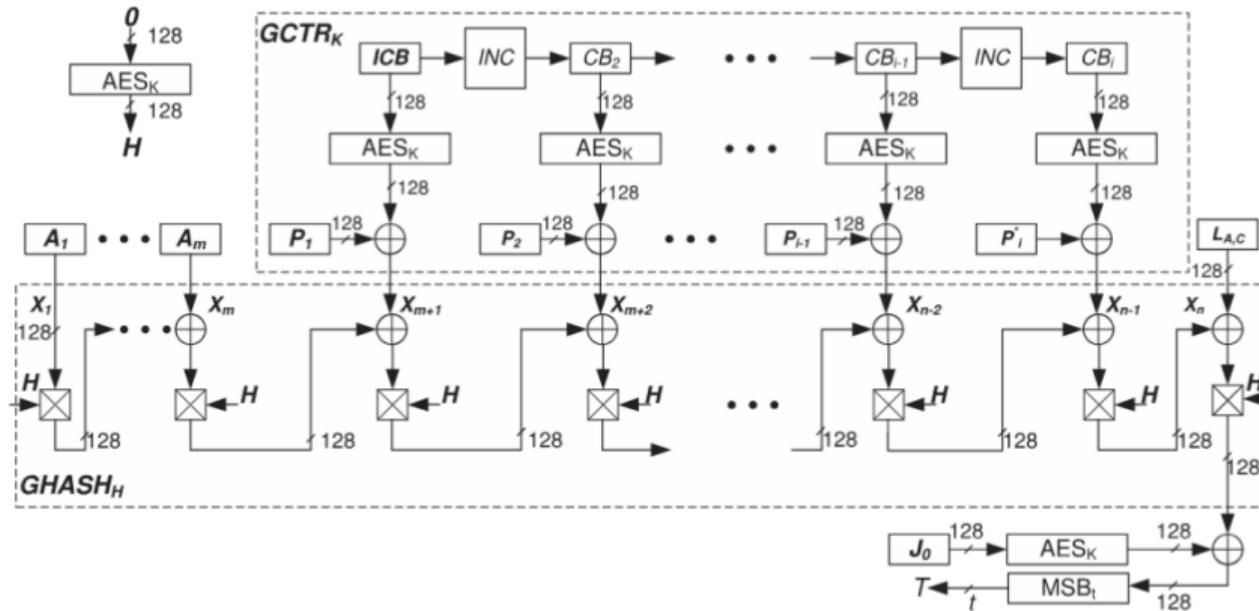
# AES – Galois/Counter Mode (GCM)

- 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  $GF(2^{128})$ .
  - $GF(2^{128})$  é o campo de Galois com 128 bits: cada bloco de 128 bits é tratado como um polinômio de grau  $\leq 127$  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



## 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:

$$b_0 + b_1x + b_2x^2 + \cdots + b_{127}x^{127}, \quad b_i \in \{0, 1\}$$

- 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](mailto:gabrielaquino@ic.ufrj.br)

Compilado em:  
September 24, 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

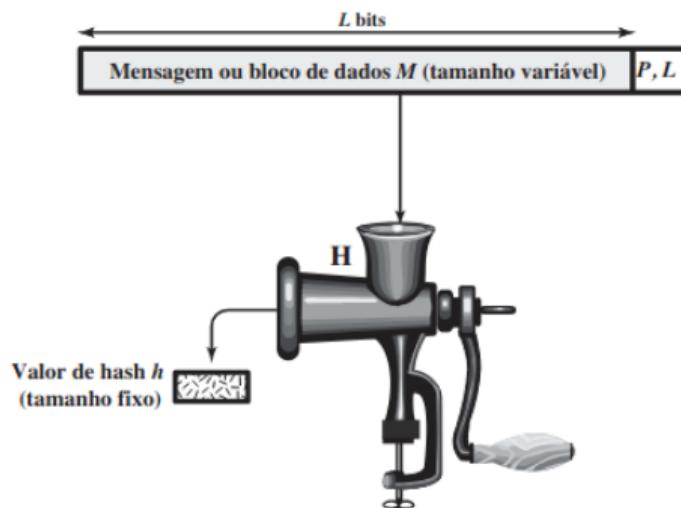
- **Mão única (one-way):** Dado um hash  $h$ , é inviável encontrar uma mensagem  $M$  tal que  $H(M) = h$ .
- **Livre de colisão (collision-free):** É inviável encontrar duas mensagens  $M_1$  e  $M_2$  tais que  $H(M_1) = H(M_2)$ .

# 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;  $h = H(M)$ .



$P, L$  = 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).

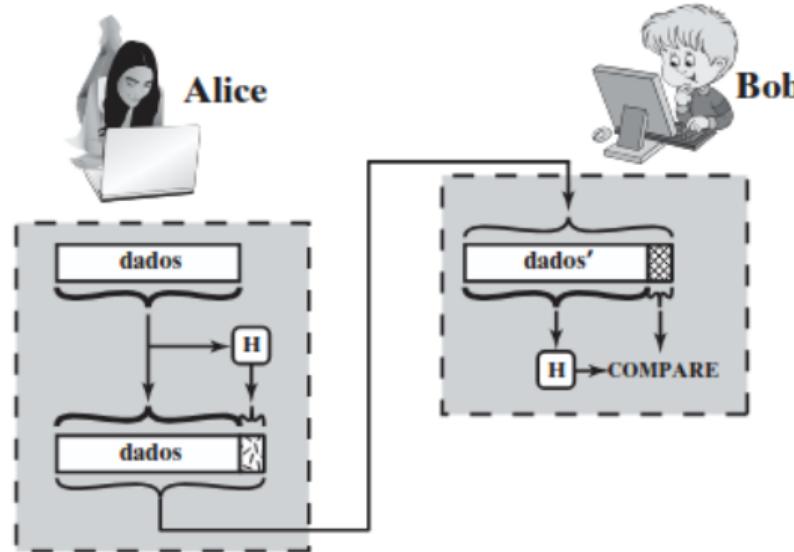
# Autenticação de mensagem

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



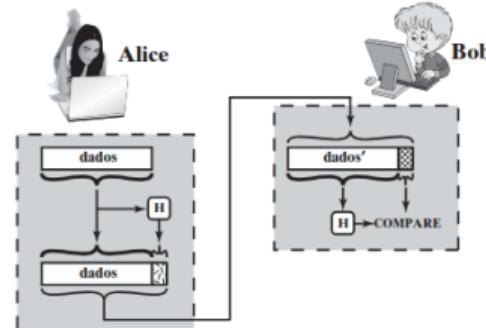
(a) Uso da função de hash para verificar integridade de dados

## Pergunta

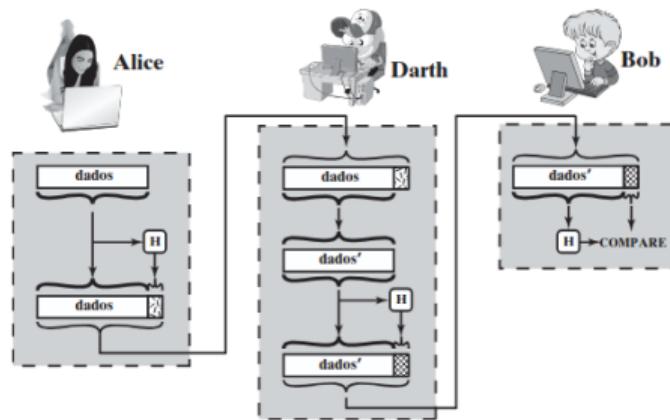
Qual o problema neste cenário?

# 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

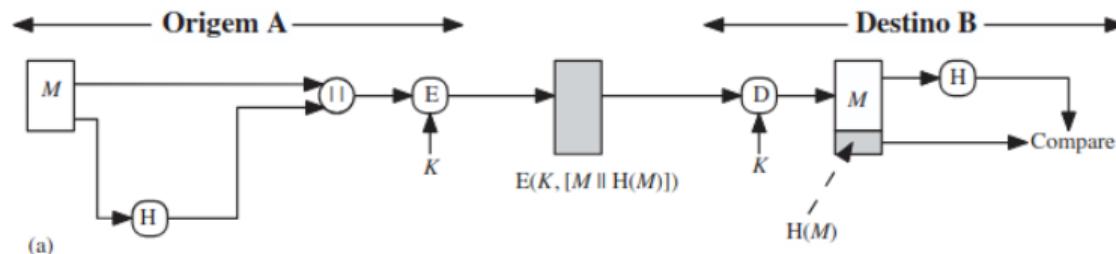
Como podemos proteger o Hash nesse cenário?

## 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

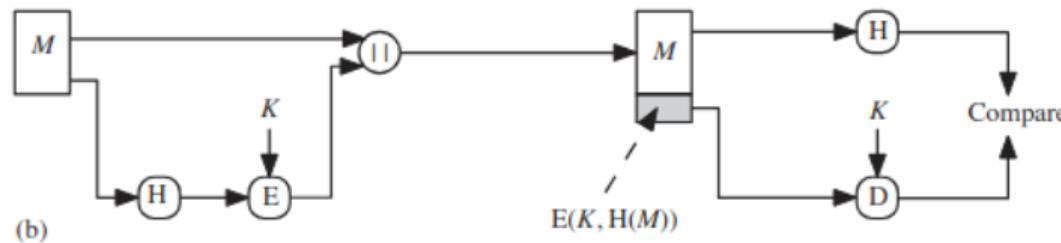
## Método A: Autenticação com hash + cifragem simétrica

- 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:

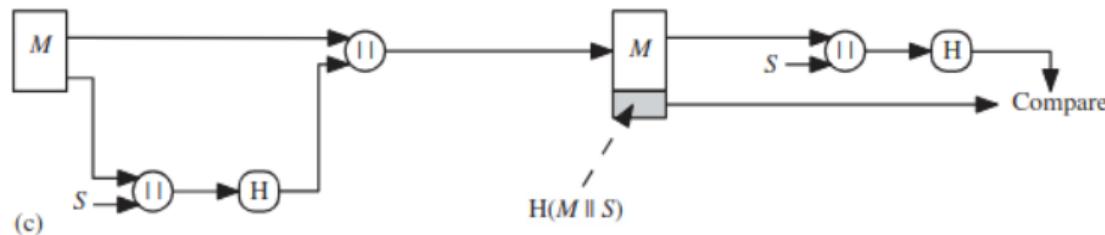
```
gpg --verify sha256sum.txt.gpg sha256sum.txt
```

## 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:  $H(M \parallel S)$ .
  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

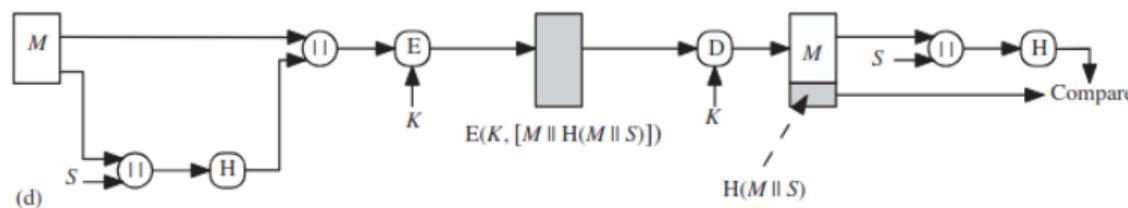
- A autenticação de mensagem normalmente é alcançada usando um **MAC** (Message Authentication Code), também chamado de função de hash chaveada.
- MACs são usados entre duas partes que compartilham uma **chave secreta** para autenticar informações trocadas.
- A função MAC recebe como entrada a chave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

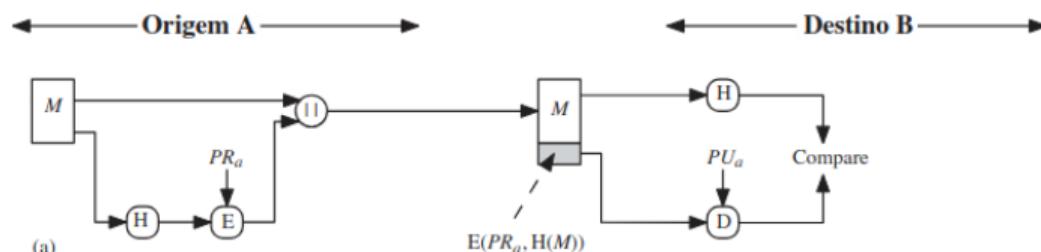


### Cenário

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

# Assinaturas Digitais

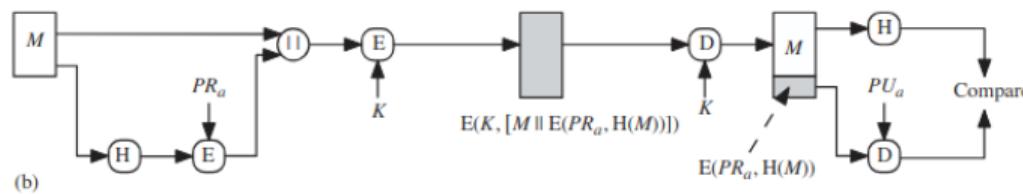
- **Usa Chave pública e privada**
- Assinatura digital é uma aplicação importante, similar à autenticação de mensagem.
- O valor de hash da mensagem é encriptado com a **chave privada** do usuário.
- Qualquer pessoa que conheça a **chave pública** do usuário pode verificar a integridade da mensagem associada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

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

onde

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

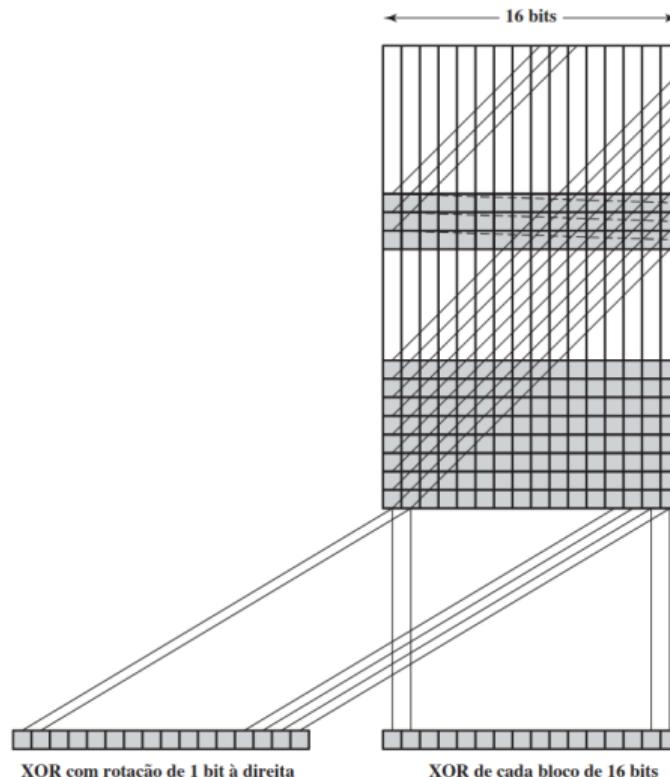
$m$  = número de blocos de  $n$  bits na entrada

$b_{ij}$  =  $i$ -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

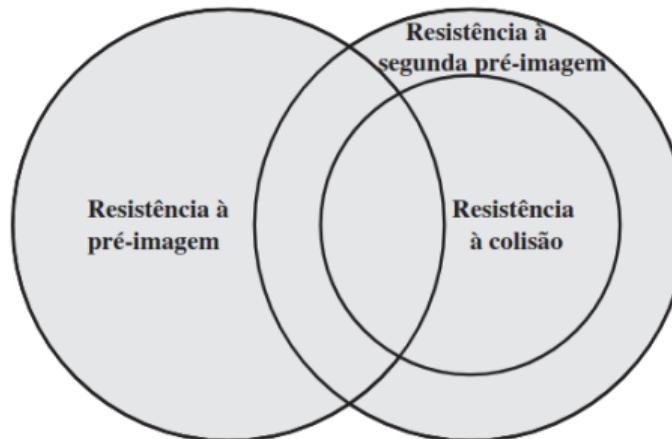
- Uma função de hash que satisfaz as primeiras cinco propriedades é chamada de **função hash fraca**.
- Se a função também satisfaz a sexta propriedade, **resistência à colisão**, é chamada de **função hash forte**.
- Função hash forte protege contra ataques onde terceiros tentam gerar mensagens diferentes com o mesmo hash.
- Exemplo de ataque sem resistência à colisão:
  - Bob cria duas mensagens diferentes com o mesmo hash ( $m_1$  e  $m_2$ ).
  - Alice assina a mensagem  $m_1$ .
  - Bob usa o hash da mensagem  $m_1$  para reivindicar que a mensagem  $m_2$  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).

**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

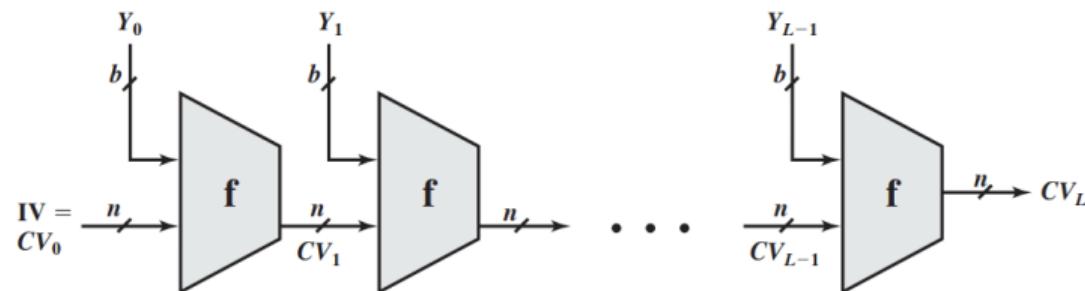
- Assim como em algoritmos de criptografia, ataques criptoanalíticos em **hashes** e **MACs** buscam explorar propriedades do algoritmo para superar a simples busca exaustiva.
- A resistência de um hash ou MAC à criptoanálise é medida comparando-se seu esforço com o esforço de um ataque por força bruta.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

A função de hash pode ser resumida da seguinte forma:

$$CV_0 = IV = \text{valor inicial de } n \text{ bits}$$

$$CV_i = f(CV_{i-1}, Y_{i-1}) \quad 1 \leq i \leq L$$

$$H(M) = CV_L$$

onde a entrada da função de hash é uma mensagem M consistindo nos blocos  $Y_0, Y_1, \dots, Y_{L-1}$

# 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.

- **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
<b>Tamanho do resumo da mensagem</b>	160	224	256	384	512
<b>Tamanho da mensagem</b>	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
<b>Tamanho do bloco</b>	512	512	512	1024	1024
<b>Tamanho da word</b>	32	32	32	64	64
<b>Número de etapas</b>	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:  $M_1, M_2, \dots, M_N$ .
- 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.