Universidade
Estadual de Londrina

Desenvolvimento de Software Seguro

Prof. Bruno Bogaz Zarpelão

Especialização em Engenharia de Software

Fundamentos Básicos

UEL

Introdução

- Por que o problema da segurança de software é importante?
- O que é a segurança de software?
- Por que o problema da segurança de software está se agravando?
- O que é *bug*, *flaw* e *defect*?

Algumas questões...

- Você acha que os softwares que temos atualmente no mercado são seguros?

Algumas questões...

- O que torna um software mais seguro?

Algumas questões...

- Quais são as ameaças à segurança de softwares que vocês conhecem?

Algumas questões...

- O que faz em seu dia a dia como desenvolvedor (ou uma função relacionada) para tornar o software em desenvolvimento mais seguro?

O problema da segurança

- Os problemas de segurança de computadores são, em sua maioria, derivados de problemas em softwares.
- A complexidade dos softwares tem aumentado.
- Softwares estão em todo lugar:
  - Computadores, smartphones, TVs, carros, aviões, usinas nucleares, etc.

Segurança de software

"Software security is about understanding software-induced security risks and how to manage them. Good software security practice leverages good software engineering practice and involves **thinking about security early in the software lifecycle**, knowing and understanding **common problems** (including language-based flaws and pitfalls), **designing for security**, and subjecting **all software artifacts to thorough objective risk analyses and testing.**" (McGraw, 2006)

Por que o problema está piorando?

- “Santíssima trindade”:
  - Conectividade
  - Extensibilidade
  - Complexidade

Conectividade

- Softwares massivamente conectados à Internet:
  - Aumenta o número de vetores de ataque.
  - Facilita a realização de ataques.
- Tendência recente (últimos 10-15 anos) de conectar software legado à Internet.
- Apenas adicionar uma VPN (*Virtual Private Network*) ao cenário não resolve o problema.

Extensibilidade

- Sistemas têm sido desenvolvidos para serem estendidos:
  - Web browsers e editores de texto, por exemplo, aceitam inúmeros plug-ins.
- Lado bom: desenvolvedores podem oferecer novas funcionalidades a usuários mais facilmente.
- Lado ruim: é relativamente comum que plug-ins maliciosos sejam encontrados.

Complexidade

- Sistemas de informação modernos são cada vez maiores e mais complexos:
  - Mais linhas de código e mais camadas levam a mais erros e problemas de segurança.
- Mesmo programas mais simples dependem de vários frameworks e bibliotecas:
  - Um “Hello World” em uma aplicação Java Web tem muito código sendo executado nos bastidores.

Exercício

- Observe o ranking da OWASP *Top 10 Web Application Security Risks* (https://owasp.org/www-project-top-ten/)e aponte:
  - a) Quais destes riscos podem ser mitigados com soluções "pós-desenvolvimento" (firewalls, testes de invasão, sistemas de detecção de intrusão, VPNs, etc.)? Por quê?
  - b) Quais destes riscos **não** podem ser mitigados com soluções "pós-desenvolvimento"? Por quê?

Exercício

- Reflita sobre os três fatores que agravam o problema da segurança de software (conectividade, extensibilidade e complexidade) e dê um exemplo que ilustre cada um deles.

Bugs, *flaws* e *defects*

- Defect: vulnerabilidades oriundas da implementação ou do design, que podem permanecer “adormecidas” por anos.
- Bug: problema na implementação do código.
  - Eles podem existir e nunca serem executados.
  - Normalmente se referem a problemas simples, que podem ser facilmente corrigidos caso detectados.

Bugs, *flaws* e *defects*

- Flaws: é um problema mais profundo, que tem origem no design do software.

Bugs, *flaws* and *defects*

- A correção de defeitos em nível de implementação (como *bugs*) às vezes não exigem que se conheça partes inteiras do programa.
- A correção de defeitos em nível de design (como *design flaws)* muitas vezes exigem que se tenha domínio de partes inteiras do programa.

Exercício

- Um sistema foi desenvolvido de forma que um atacante consegue passar comandos SQL em um formulário de cadastro de pessoa física. Temos um caso de *bug* ou *flaw*?

Exercício

- Em um programa em C, foi descoberto o uso da função insegura *gets()*, que possibilita a execução de ataques de buffer overflow. Neste caso temos um *bug* ou *flaw*?

Exercício

- Em um sistema desenvolvido em sua empresa, um cliente que esqueceu a sua senha pode pedir que uma nova senha seja enviada para seu e-mail. O problema é que o próprio usuário insere o e-mail, sem que haja qualquer validação sobre a sua autenticidade. Temos um *bug* ou *flaw*?

Exercício

- A sua empresa desenvolveu um sistema para condomínios. Neste sistema, temos um módulo de agendamento de uso de espaços compartilhados. O problema é que moradores do condomínio conseguem não só visualizar, mas também apagar os agendamentos realizados por outros moradores. Temos um *bug* ou *flaw*?

Criptografia e Segurança de Redes

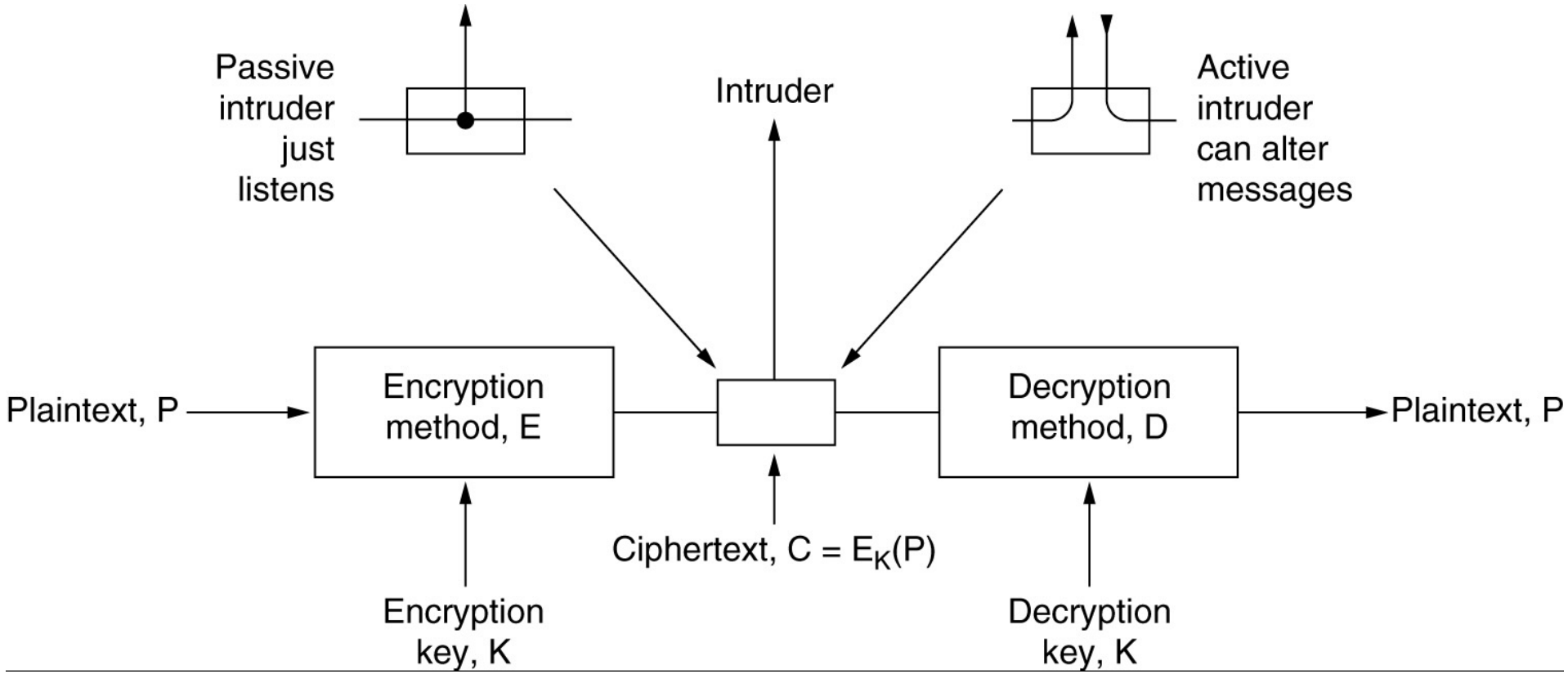
Introdução

- Como funcionam as cifras simétricas de bloco?
- Como devemos executar as cifras simétricas?
- Como podemos evitar o compartilhamento inseguro de chaves?
- O que são as funções hash?
- Como funcionam as assinaturas digitais e os certificados?

Visão geral da criptografia

- Não está ligada necessariamente à segurança de computadores. É um conceito mais amplo e antigo.
- Historicamente, militares e diplomatas estão entre os que mais contribuíram para a criptografia.

Visão geral da criptografia



Visão geral da criptografia

- Não podemos confiar a segurança com base no sigilo do método de criptografia.
- Esse método pode ser descoberto e a segurança acabará.
- A segurança está em cima da chave.
- A chave pode ser mudada facilmente caso seja quebrada. O método, muitas vezes, não.

Visão geral da criptografia

- Dois princípios importantes:
  - Redundância: inserir bytes a mais na mensagem criptografada para dificultar a vida do atacante.
  - Atualidade: garantir que a mensagem é sempre recente, evitando que o atacante mande mensagens antigas repetidas nos enganando.

Visão geral da criptografia

- Alguns objetivos da criptografia:
  - Autenticação do emissor.
  - Confidencialidade.
  - Integridade da mensagem.
  - Não repúdio.

Importante

- Criptografia não é garantia de segurança:
  - Bugs em softwares.
  - Engenharia social, etc.
- Não importa se está usando o melhor algoritmo com a maior chave, sempre pode haver um elo fraco.

Importante

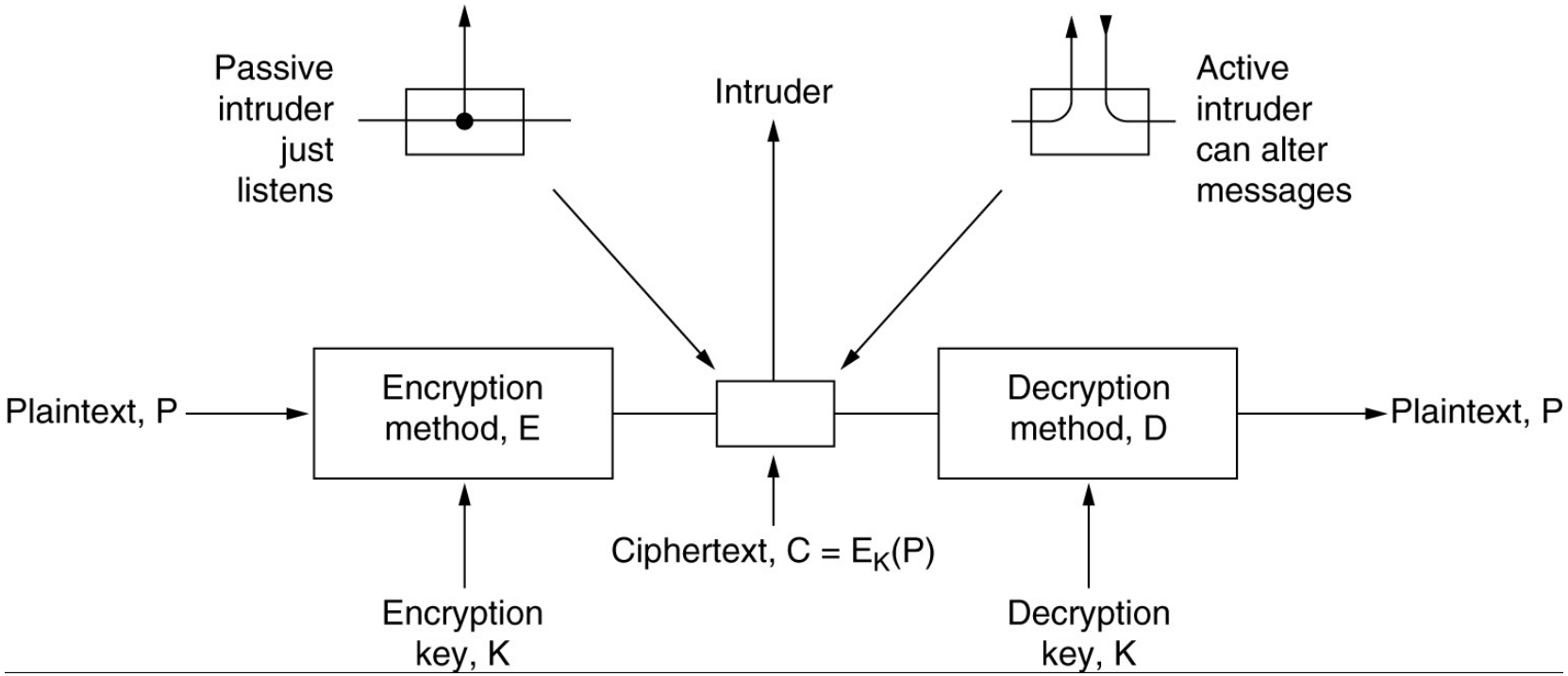
- É importante conhecer pelo menos as noções básicas de criptografia.
- Todos na área de TI tem algum contato com criptografia.
- Ferramentas não são muito amigáveis e podem induzir a erro.

Exercício

- Aponte duas situações do seu cotidiano onde a criptografia é utilizada.

Cifras simétricas

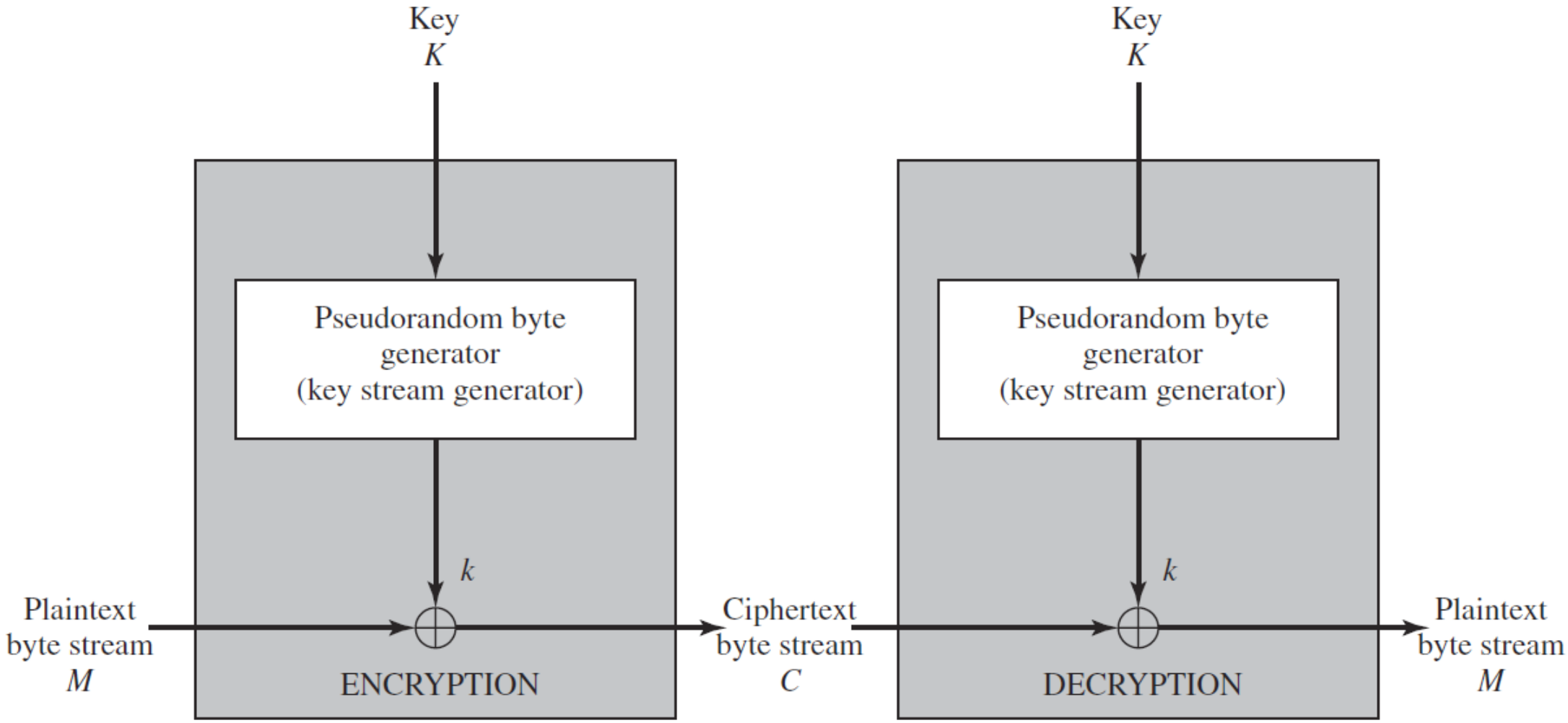
- A mesma chave é usada para cifrar e decifrar a mensagem;



Cifras simétricas de fluxo

- É um algoritmo de criptografia simétrica no qual:
  - A entrada é um fluxo contendo o texto plano.
  - A saída é um texto cifrado produzido byte a byte.
- O RC4 é um dos exemplos mais famosos (já considerado inseguro). ChaCha20 é outro exemplo.

Cifra de Fluxo



UEL

Cifras simétricas de bloco

- As cifras de bloco operam sobre blocos de texto plano com *n* bits para produzir como saída blocos de texto cifrado do **mesmo tamanho**.
- A chave usada para cifrar e decifrar é a mesma (cifra simétrica).
- Tipicamente, um bloco tem 64 ou 128 bits.
- A grande maioria dos algoritmos de criptografia usados em redes de computadores são baseados em cifras de bloco.

Cifras simétricas de bloco

- A ideia é realizar permutações e substituições nos bits presentes em um bloco.
- Estas permutações e substituições dependem da chave.
- Um dos exemplos mais conhecidos: AES (Advanced Encryption Standard).
- O AES trabalha com blocos de 128 bits e aceita chaves de 128, 192 e 256 bits.

Exercício

Cifrar arquivo usando a cifra simétrica AES (Advanced Encryption Standard):

1- Crie um arquivo texto com o seu nome e salve o arquivo

2- Crie uma chave secreta e a converta para hexadecimal.

3- Execute o seguinte comando: *openssl enc –aes-128-ecb -e –k <chave em hexadecimal> -in <nome do arquivo> -out cipher.bin*

4- Abra o arquivo cipher.bin para ver o conteúdo cifrado.

Exercício

- Cifre novamente o mesmo arquivo, mas agora mude a chave em apenas um dígito hexadecimal.
- Tome o cuidado de gravar o resultado da cifragem em um arquivo diferente do gerado no exercício anterior.
- Compare este arquivo gerado agora com o arquivo anterior. Mudamos apenas um dígito hexadecimal e a entrada é a mesma. **As saídas são parecidas?**
- **Dica: abra os arquivos com o conteúdo cifrado utilizando o hexdump.**

UEL

Exercício

Decifrar arquivo usando a cifra simétrica AES (Advanced Encryption Standard):

1- Para decifrar o arquivo, execute o comando: *openssl enc –aes-128-ecb -d -in <param 1> -out teste.dec.txt –K <param 2>*

4- Abra o arquivo teste.dec.txt para ver o conteúdo decifrado e verificar se ele é o mesmo do conteúdo original.

Modos de operação

- Suponha que você vai fazer um programa que usa uma biblioteca para cifrar um documento de 1MB usando o AES. Esta biblioteca lhe oferece duas funções: `encrypt(textblock, keyword)` e `decrypt(textblock,keyword)`. Descreva em linhas gerais como você faria esta implementação, considerando que a biblioteca oferece apenas a possibilidade de cifrar e decifrar bloco a bloco.
- Agora pense: será que esta maneira é segura?

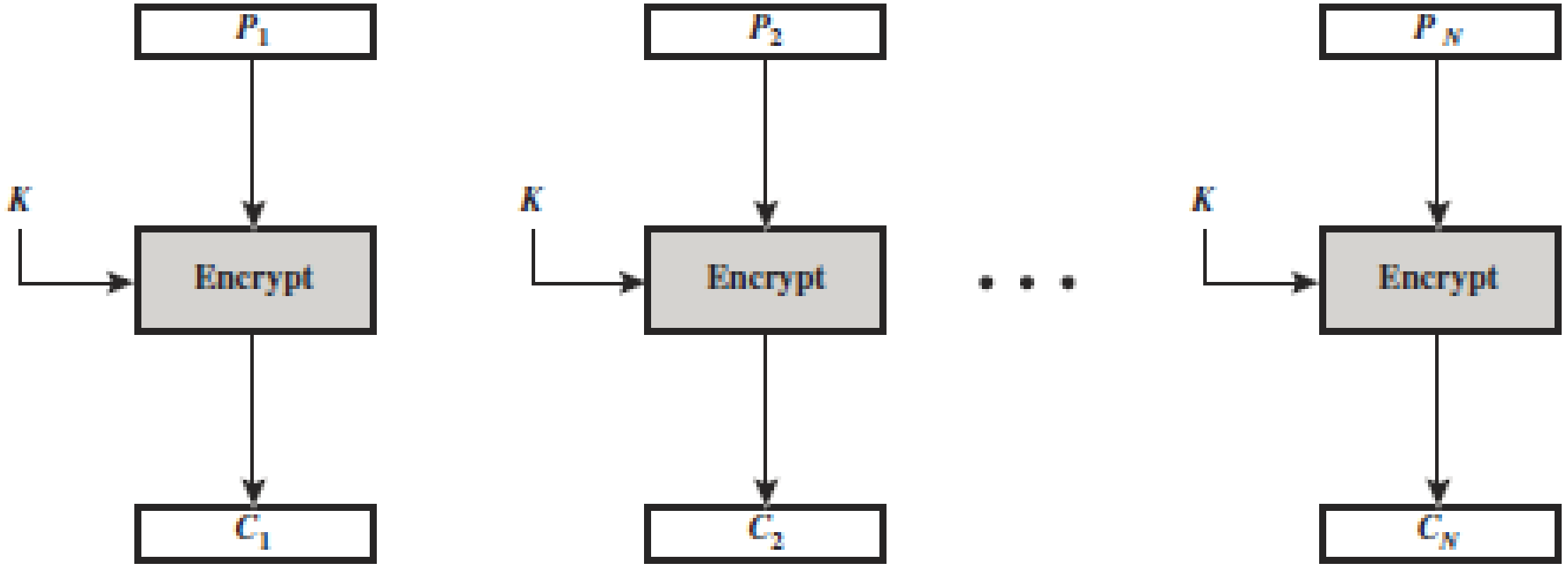
Modos de operação

- Um modo de operação é uma técnica para melhorar o efeito de um algoritmo de criptografia ou para adaptar um algoritmo para uma aplicação, por exemplo, aplicar um algoritmo de cifra de bloco sobre um fluxo contínuo (*stream)* de dados.

Electronic Codebook Mode (ECB)

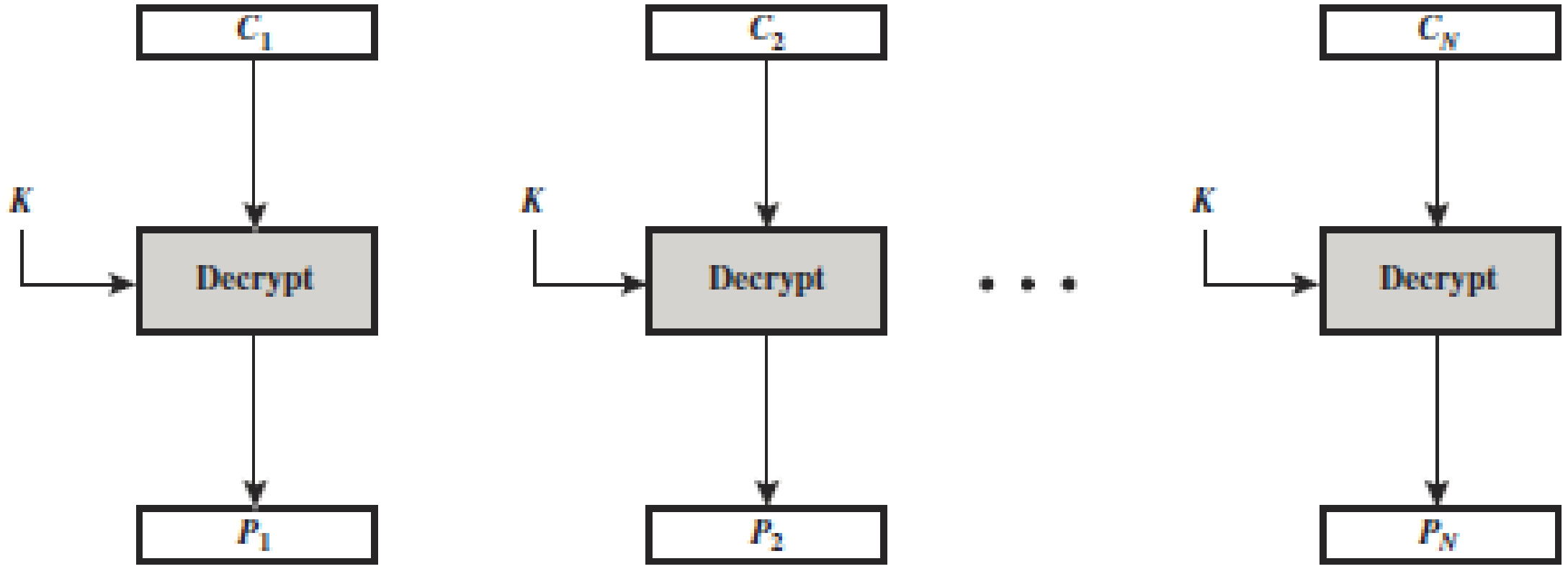
- É o modo de operação mais simples de todos.
- A mesma chave é aplicada para os diferentes blocos de texto plano.
- Blocos de texto plano **iguais** produzirão blocos cifrados **iguais**.

Electronic Codebook Mode (ECB)



(a) Encryption

Electronic Codebook Mode (ECB)



(b) Decryption

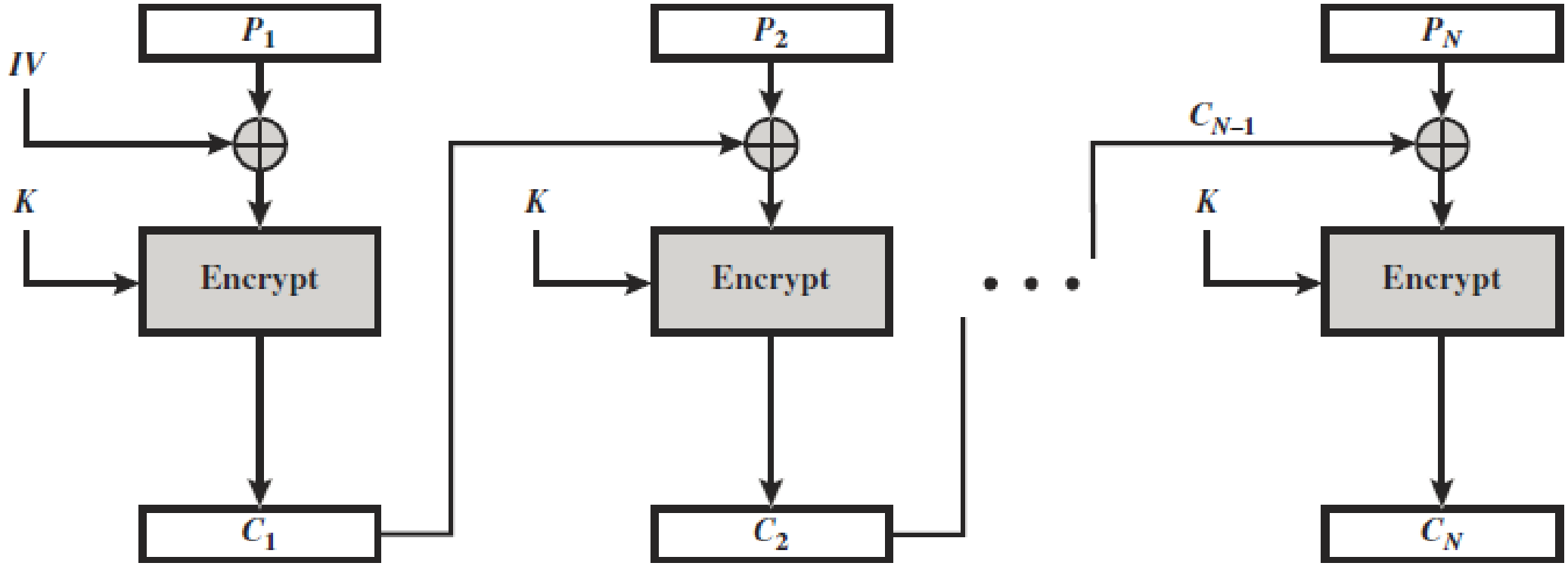
Electronic Codebook Mode (ECB)

- Deve ser utilizado apenas para cifragem de mensagens pequenas (por exemplo, chaves secretas).
- Em mensagens grandes, a repetição de trechos pode facilitar a criptoanálise.

Cipher Block Chaining Mode (CBC)

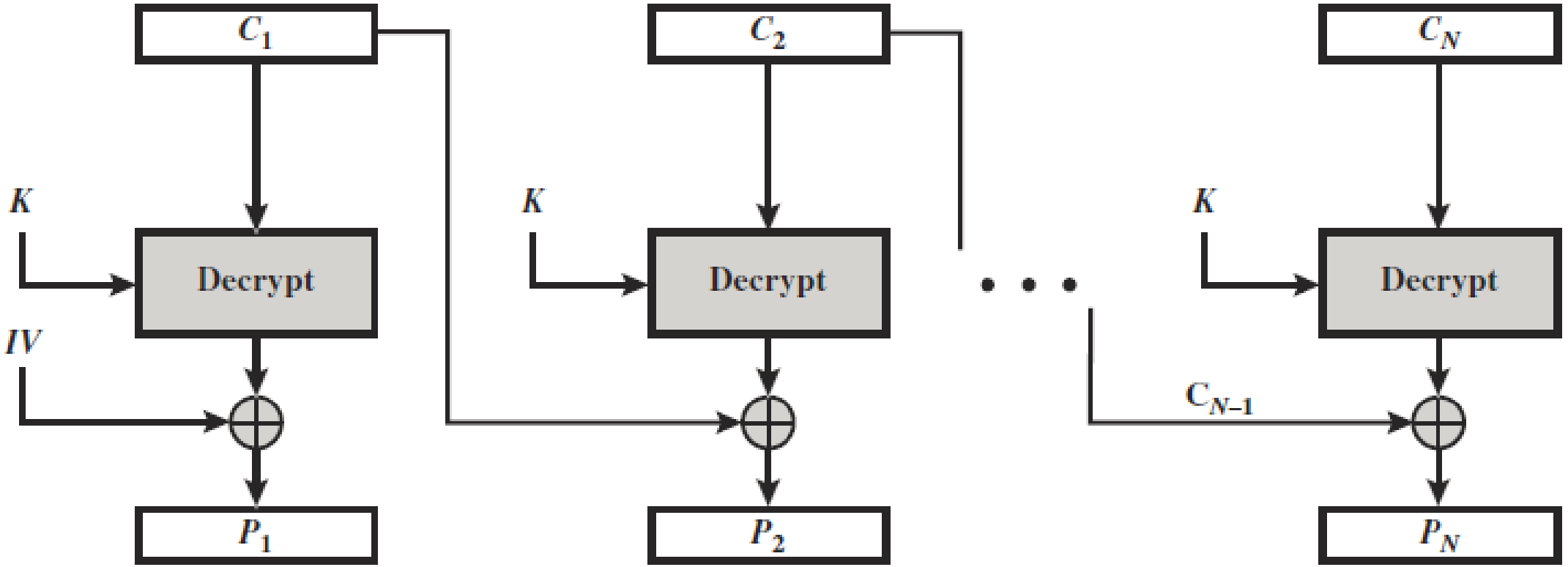
- Blocos de texto plano iguais produzem blocos cifrados diferentes.
- O algoritmo de criptografia recebe o resultado da operação XOR entre o bloco de texto plano atual e o bloco cifrado anterior.
- A mesma chave é usada o tempo todo.
- É necessário um vetor de inicialização (IV).

Cipher Block Chaining Mode (CBC)



(a) Encryption

Cipher Block Chaining Mode (CBC)



(b) Decryption

Cipher Block Chaining Mode (CBC)

- O IV deve ser secreto e não pode ser previsível para o atacante.
- Dois métodos para gerar um IV:
  - Aplicar a cifra de bloco sobre um *nonce* com a mesma chave utilizada no restante do processo. O *nonce* deve ser único para cada operação de criptografia (um timestamp, um contador, uma combinação deles, etc.)
  - Gerar um bloco aleatório com um gerador de números aleatórios.

Gerador de números pseudo-aleatórios

- Processo determinístico que recebe “sementes” como entrada e provê como saída uma serie de números que parecem aleatórios de acordo com testes estatísticos específicos.
- Necessidade frequente em criptografia.
- **Cuidado:** nem toda função de geração de números aleatórios tem segurança criptográfica.

Modos de operação

- O NIST definiu 5 modos de operação no documento SP 800-38A:
  - ECB (Electronic Codebook), CBC (Cipher Block Chaining), CFB (Cipher Feedback), OFB (Output Feedback) e CTR (Counter).
- Esses modos de operação são indicados para uso com cifras de bloco simétricas.
- Detalhe importante: em geral, se o último bloco não tiver o tamanho exigido, ele deve ser completado em uma operação chamada *padding*.

Exercício

Cifrar arquivo usando a cifra simétrica AES (Advanced Encryption Standard):

1- Crie um arquivo texto com o seu nome e salve o arquivo

2- Crie uma chave secreta e a converta para hexadecimal.

3- Execute o seguinte comando: *openssl enc –aes-128-cbc -e –k <chave em hexadecimal> -in <nome do arquivo> -out cipher.bin -iv <iv em hexadecimal>*

4- Abra o arquivo cipher.bin para ver o conteúdo cifrado.

Exercício

Decifrar arquivo usando a cifra simétrica AES (Advanced Encryption Standard):

1- Para decifrar o arquivo, execute o comando: *openssl enc –aes-128-cbc -d -in <param 1> -out teste.dec.txt –k <param 2> -iv <param3>*

4- Abra o arquivo teste.dec.txt para ver o conteúdo decifrado e verificar se ele é o mesmo do conteúdo original.

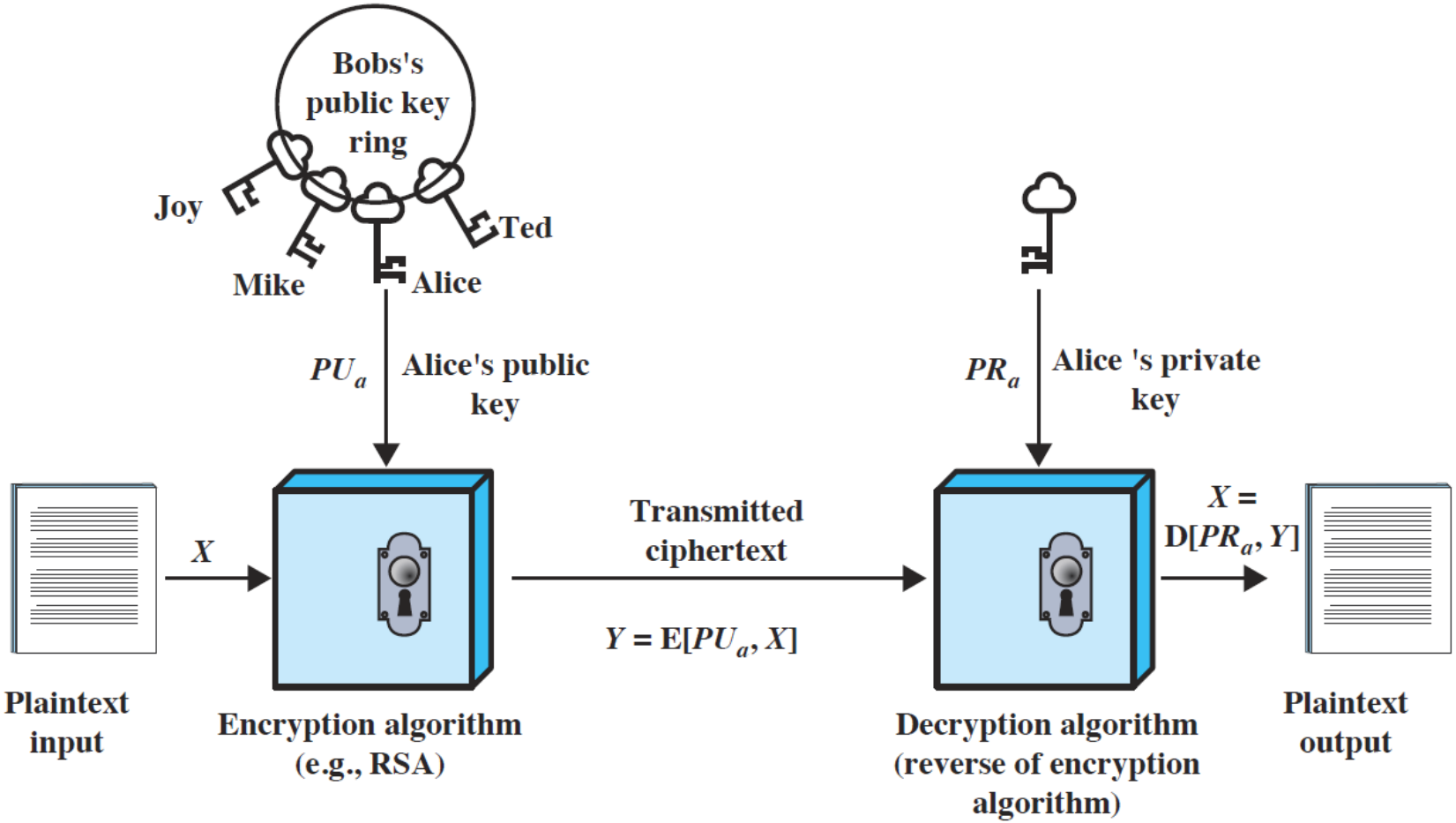
Exercício

- Com a pandemia de COVID-19, grande parte do mundo tem trabalhado de casa e a ferramenta de vídeo conferência Zoom tem sido frequentemente utilizada. No início, essa ferramenta criptografava as comunicações utilizando o AES-128 no modo de operação ECB. Explique qual é o problema nessa situação.

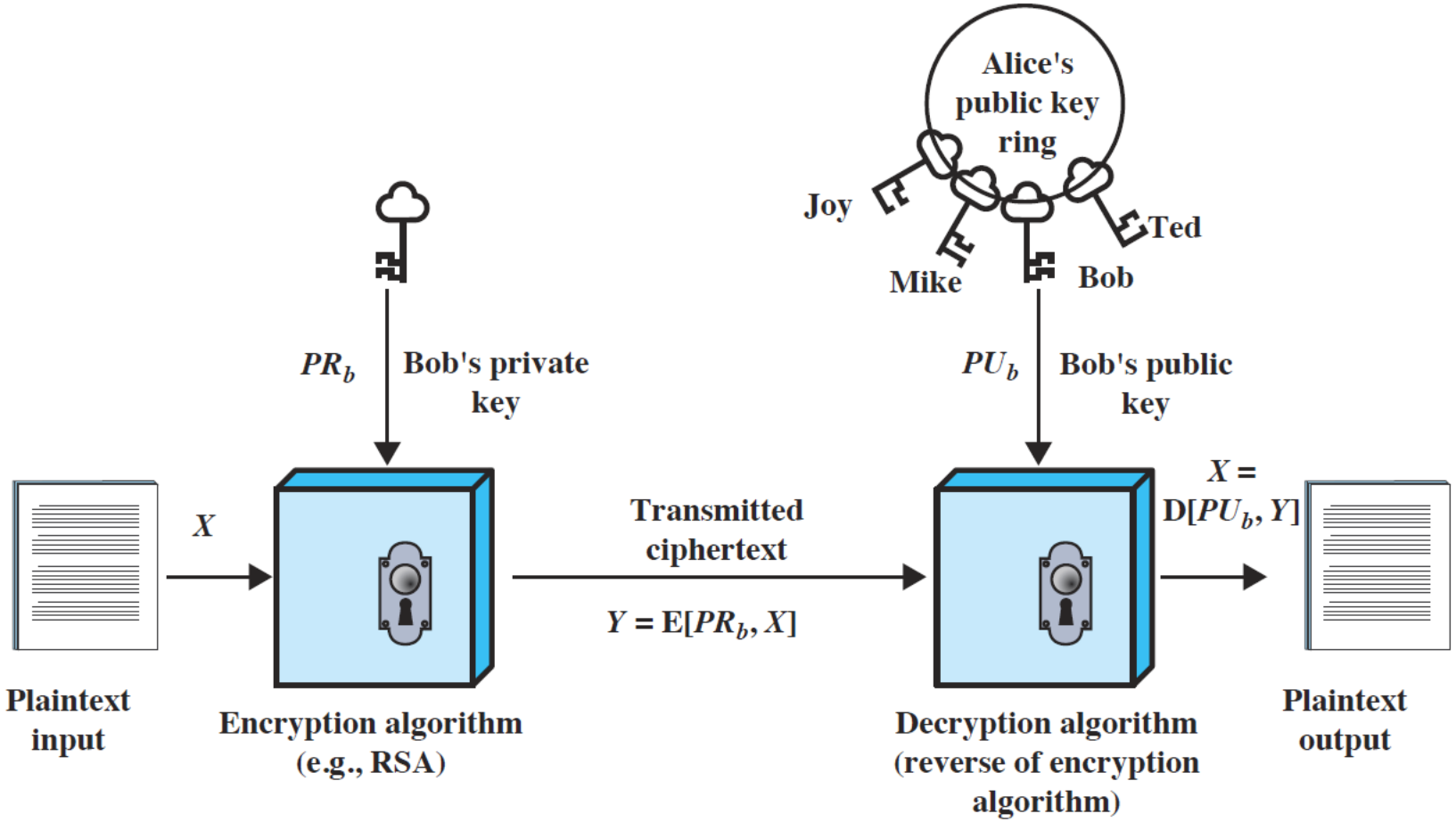
Criptografia de chave pública

- As cifras simétricas têm um ponto crítico: compartilhamento da chave secreta.
- A solução pode ser a criptografia de chave pública, baseada em duas chaves: pública e privada.
- As cifras assimétricas foram propostas por Diffie e Hellman em 1976.
- Além da confidencialidade, resolvem outro problema: permitem autenticação por meio de assinatura digital.

Criptografia de chave pública



Criptografia de chave pública



Criptografia de chave pública

- Os algoritmos de chave pública devem possuir a seguinte característica:
  - é computacionalmente inviável descobrir a chave para decifragem conhecendo apenas o algoritmo e a chave para cifragem.
- Uma característica adicional importante:
  - Qualquer uma das chaves pode ser utilizada para cifrar, ficando a outra para decifrar.

Criptografia de chave pública

- Ela é de 60 a 70 vezes mais lenta que a criptografia simétrica.
- O RSA (Rivest, Shamir, Adleman) é um exemplo.

RSA

- Rivest, Shamir e Adleman (MIT) receberam o ACM Turing Award de 2002 pelo RSA.
- Exige chaves de, pelo menos, 1024 bits, o que o torna mais lento.
- Baseia-se no fato que é inviável fatorar um número resultante da multiplicação de dois primos muito grandes.

Funções Hash

- Na área de segurança de redes, temos as funções hash criptográficas.

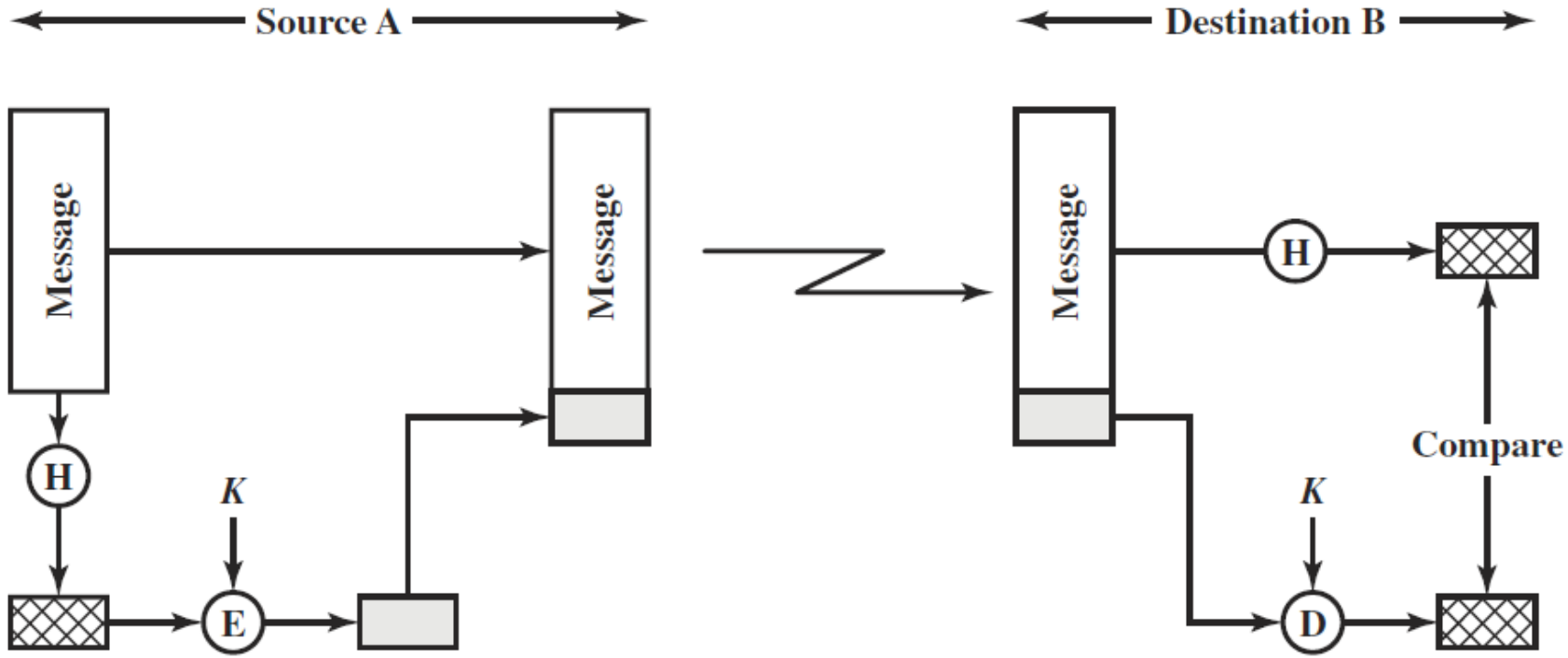
Requisitos

- Uma função hash criptográfica deve respeitar os seguintes requisitos:
  - Aceitar mensagens de entrada de tamanho variável.
  - Retornar valores de tamanho fixo.
  - Deve ser eficiente em termos de complexidade computacional.
  - Dado qualquer valor h, deve ser computacionalmente inviável encontrar o M tal que $h = H(M)$.

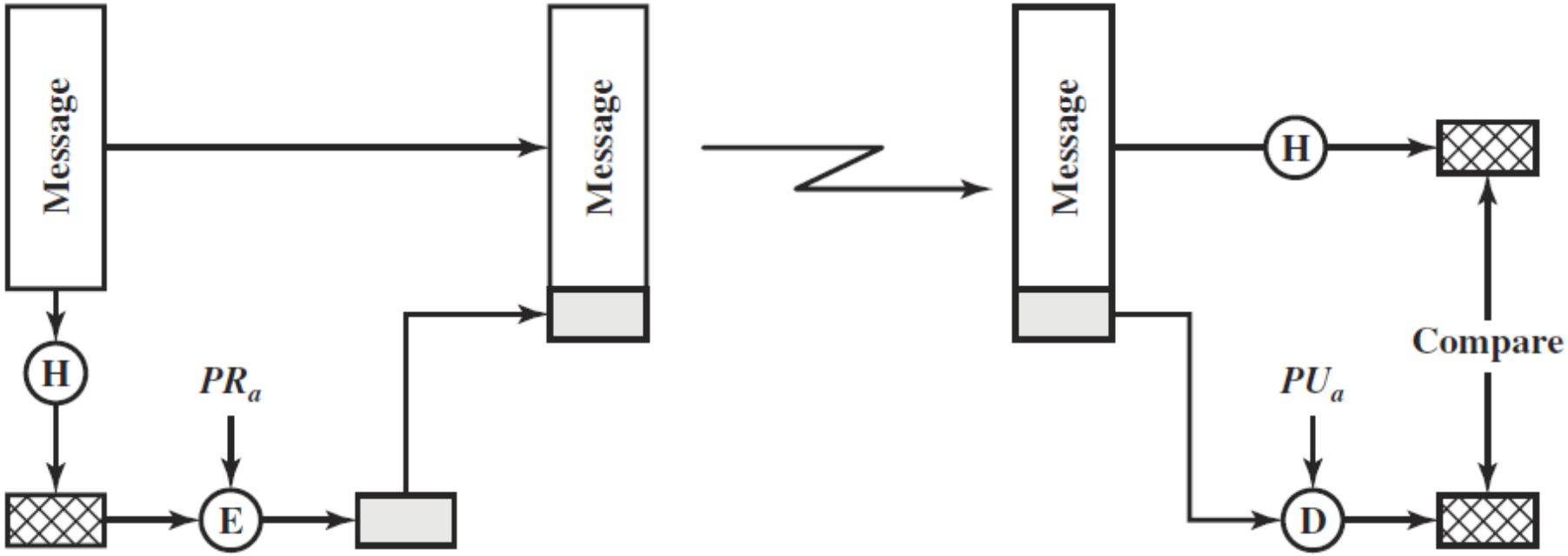
Requisitos

- Uma função hash criptográfica deve respeitar os seguintes requisitos:
  - Dado H(*M*), deve ser computacionalmente inviável encontrar uma mensagem *N* que implique em H(*N*) = H(*M*).
  - Deve ser computacionalmente inviável encontrar duas mensagens *M* e *N* para as quais H(*M*) = H(*N*).
  - A saída de H deve ser pseudoaleatória.

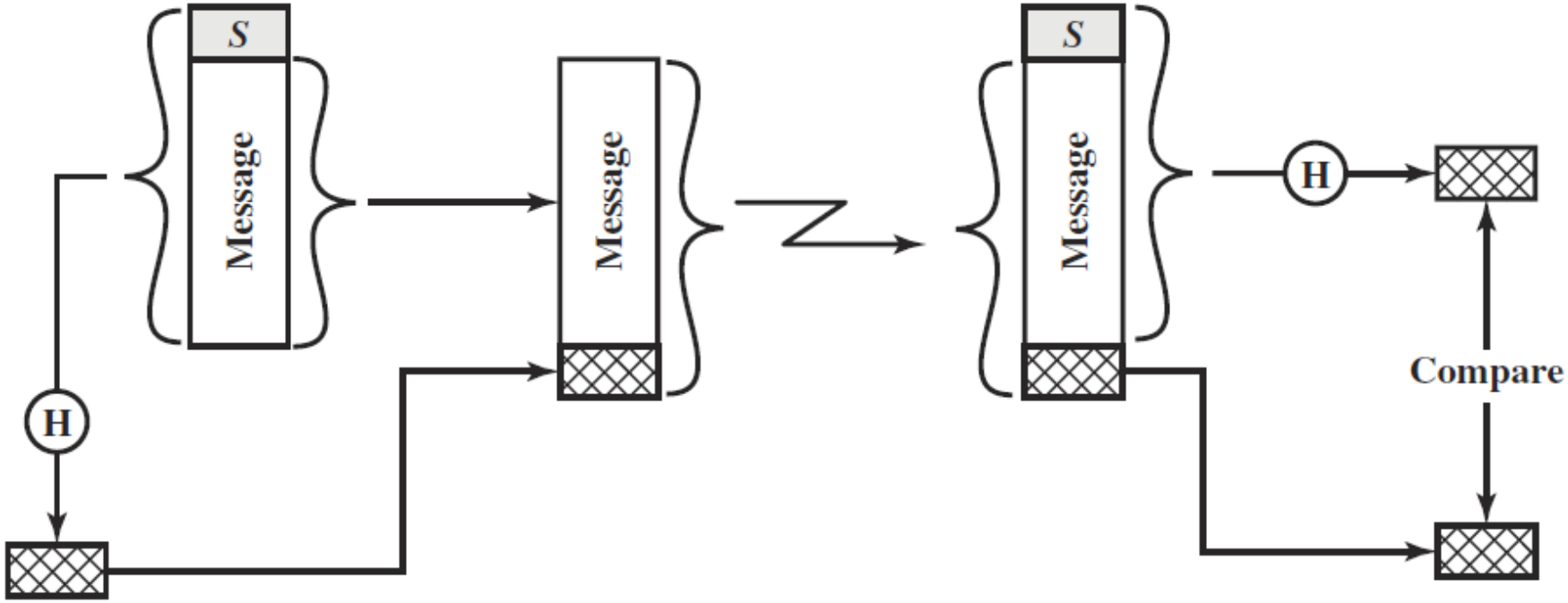
Autenticação usando função hash



Autenticação usando função hash



Autenticação usando a função hash



Segurança das funções hash

- A força de uma função hash contra ataques de força bruta depende do tamanho *n* do código hash de saída.

Exercício

- Quais são as características necessárias a uma função hash criptográfica?

SHA

- A função hash mais utilizada nos últimos anos é o SHA (Security Hash Algorithm).
- Foi desenvolvida pelo NIST e padronizada no documento Secure Hash Standard (FIPS 180).
- O SHA-0, a primeira versão, foi substituída pelo SHA-1 em 1995 quando foram descobertas suas primeiras vulnerabilidades.

SHA

- O SHA-2 acaba sendo internamente parecido com os seus antecessores, o que pode representar uma fraqueza.
- O NIST lançou um concurso para criar uma nova versão do SHA.
- Recentemente, foi lançado o SHA-3, que produz saídas nos seguintes tamanhos: 224 (SHA3-284) bits, 256 bits (SHA3-256), 384 bits (SHA3-384) e 512 bits (SHA3-512).

Exercício

Acesse o site https://codebeautify.org/sha3-512-hash-generator ou https://www.cryptool.org/en/cto/openssl e faça o seguinte:

a) Gere o código hash para duas palavras bem semelhantes. Os resultados são parecidos?

b) Gere o código hash para uma palavra e para um texto de tamanho maior. É possível dizer qual código hash corresponde a qual entrada?

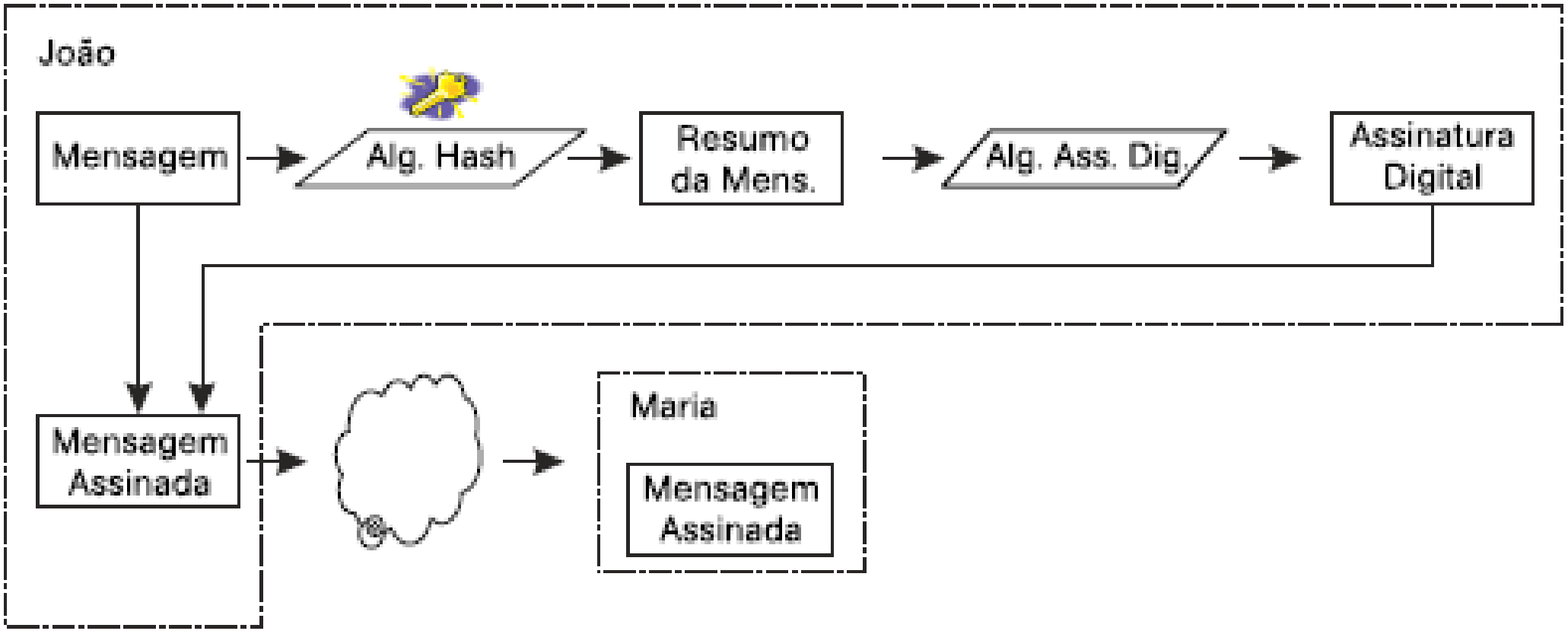
Assinaturas digitais

- Fazer assinaturas digitais que substituam as assinaturas reais. Dificuldades:
  - O receptor deve verificar a identidade alegada pelo transmissor.
  - O transmissor não pode repudiar o conteúdo da mensagem.
  - O receptor não pode inventar a mensagem e responsabilizar o transmissor.

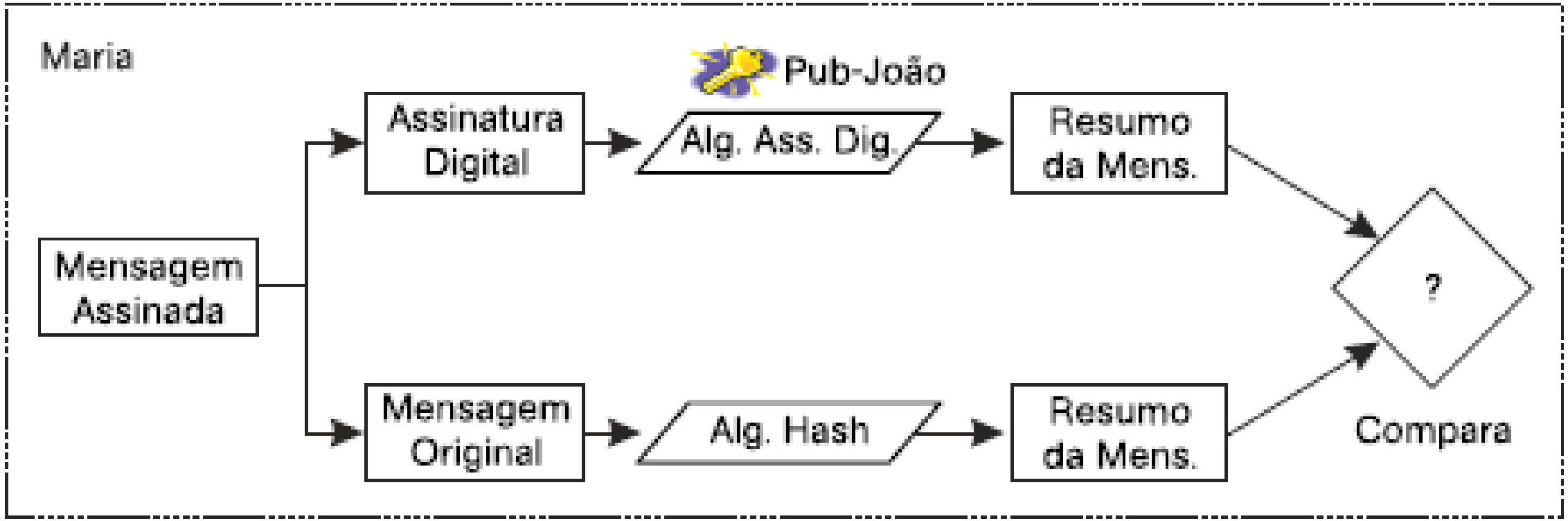
Assinaturas digitais

- As chaves públicas são normalmente utilizadas para resolver o problema de assinaturas digitais.

Processo de assinatura digital



Processo de assinatura digital



Divulgação de chaves públicas

- A natureza das chave públicas faz com que seja importante divulgá-las amplamente.
- Indivíduos podem anunciar suas chaves livremente:
  - Problema: como garantir que aquela chave pública realmente pertence a uma determinada entidade?

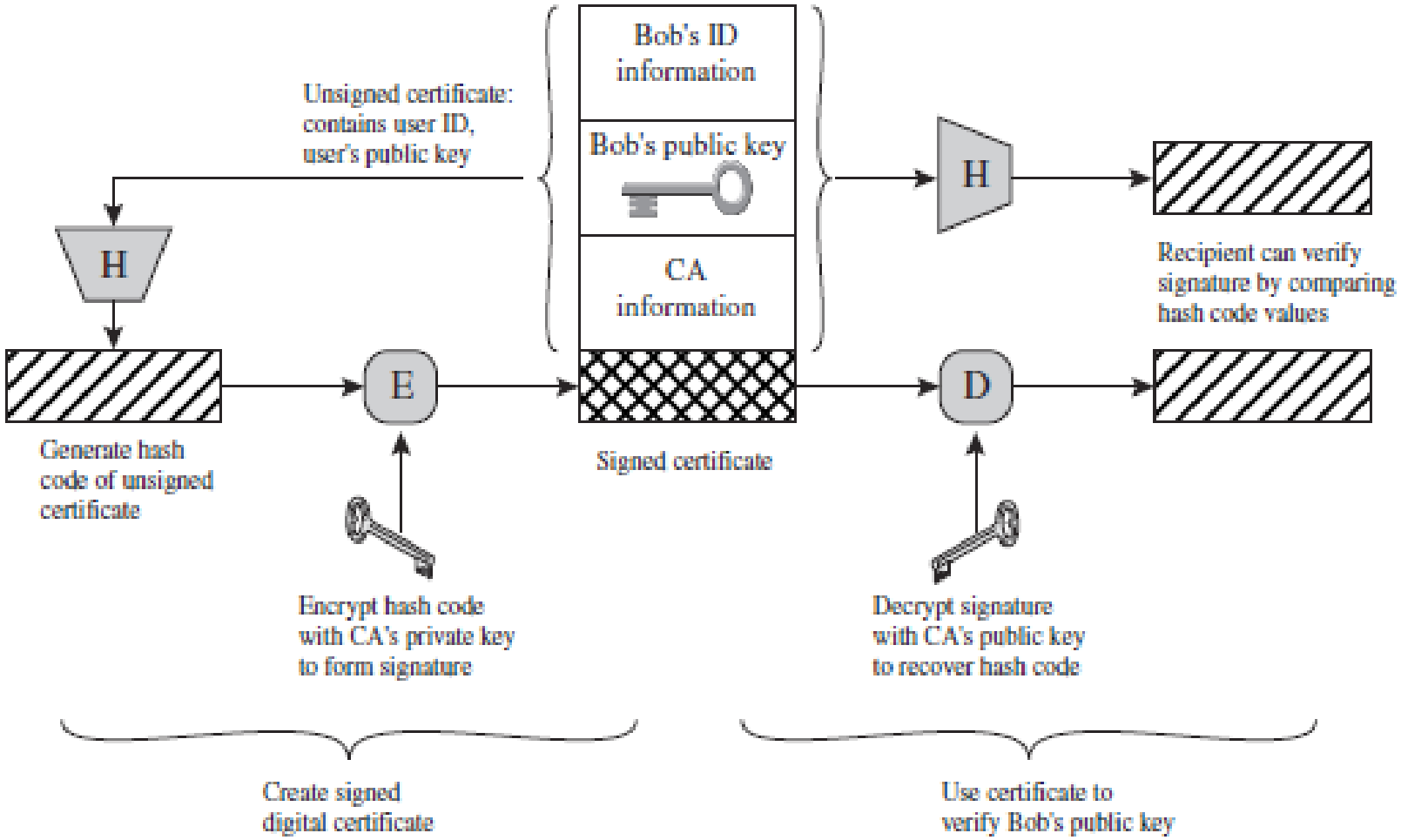
Certificados digitais

- Certificados digitais utilizam a criptografia de chave pública (criptografia assimétrica).
- Definição rápida de certificado digital:
  - Chave pública de um usuário ou sistema que é assinada digitalmente por uma autoridade certificadora (CA) usando a chave privada dela.

Certificado digital

- Um certificado digital pode conter diversas informações:
  - Nome, endereço e empresa do solicitante.
  - Chave pública do solicitante.
  - Validade do certificado (muito importante).
  - Nome e endereço da autoridade certificadora.
  - Políticas de utilização.

Certificado Digital



Autoridades Certificadoras

- As Autoridades Certificadoras (CA) têm a função de criar, manter e controlar todos os certificados que elas emitem.
- Elas devem, inclusive, invalidar certificados expirados ou comprometidos.
- O gerenciamento de certificados digitais é um assunto complexo e delicado.

Infra-estrutura de chave pública

- PKI: Public Key Infrastructure.
- É uma união de pessoas, organizações, hardware, software, políticas e procedimentos para prover a infraestrutura necessária ao uso adequado de certificados digitais baseados em chaves públicas.
- Exemplo: ICP-Brasil.
- Uma PKI oferece confiabilidade nas transações que utilizam certificado digital.

Funções da PKI

- Registro: uma entidade se registra em uma CA por meio de uma Autoridade de Registro (RA – Registration Authority).
- Certificação: a CA envia o certificado digital para a entidade que requereu e o coloca em um repositório (diretório).
- Recuperação do par de chaves: a CA pode guardar uma cópia de segurança da chave privada da entidade e fornecê-la em caso de necessidade.

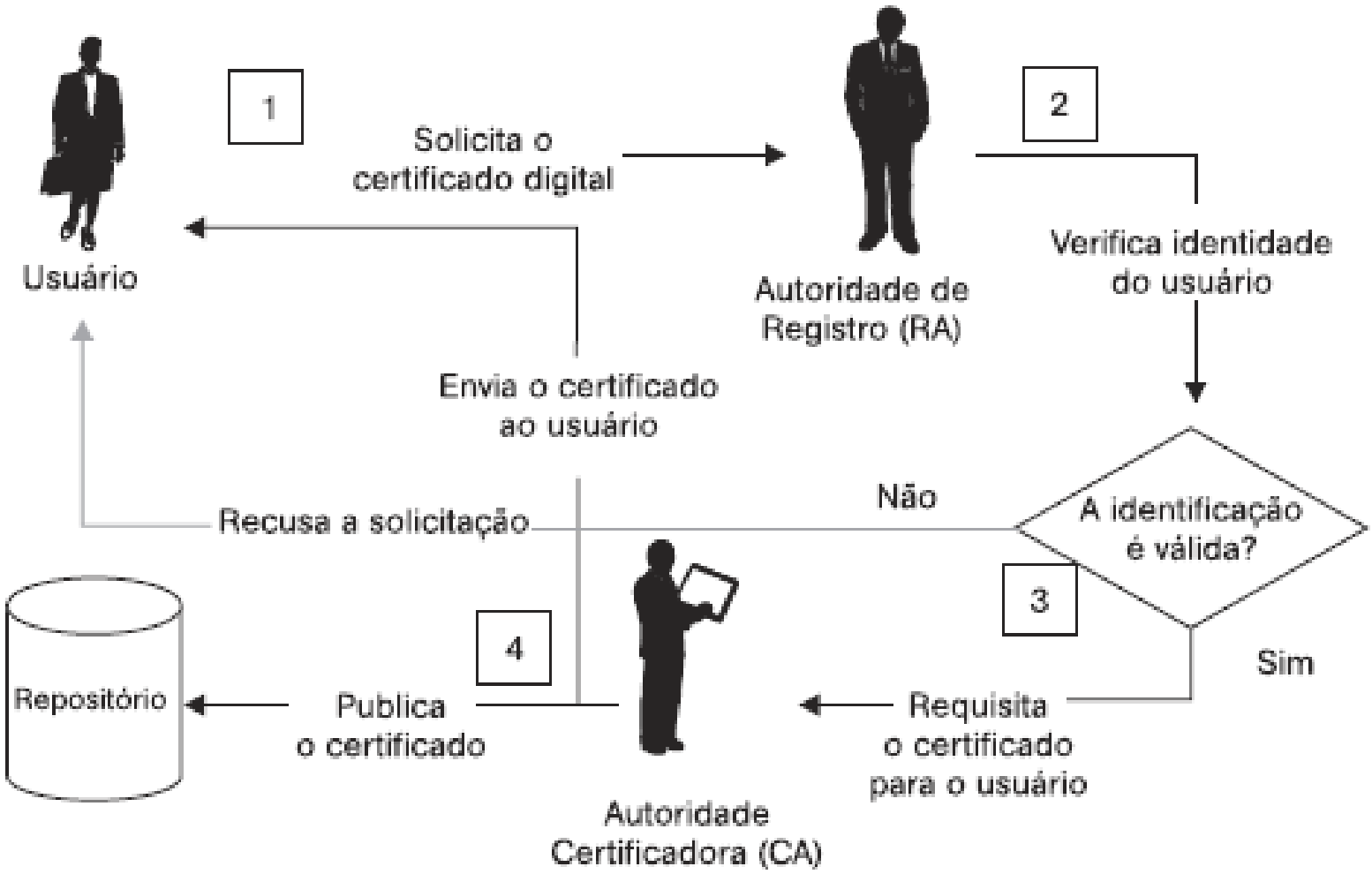
Funções da PKI

- Geração de chaves: pode ser feita pelo próprio usuário ou pela CA.
- Atualização das chaves: quando a chave é comprometida ou expira.
- Revogação: o certificado por de ser revogado em caso de comprometimento do certificado, mudanças no nome da entidade, etc. Certificados revogados devem ser incluídos em uma lista pública.

Funções da PKI

- Distribuição e publicação de certificados e de notificações de revogação.

Solicitação de certificado digital



Exercício

- Acesse o site: https://letsencrypt.org/how-it-works/
- Leia o conteúdo da página e explique de maneira resumida como funciona o processo de emissão de um certificado pela Let´s Encrypt.
- Vocês acham isso seguro?
- Esse processo seria possível pra qualquer tipo de certificado?

Exercício

- Implementar o laboratório “Crypto Lab – Public Key Cryptography and PKI”

Abuse and Misuse Cases

Introdução

- Como podemos trabalhar com a parte de segurança já no levantamento de requisitos?

Requisitos

- Ponto inicial do desenvolvimento de um software: levantar requisitos.
- Quão importante é definir bem os requisitos?
- Como trabalhamos hoje com o levantamento e definição de requisitos?

Algumas características de requisitos

- Definem o que o sistema deve (e não deve) fazer.
- Definem quem interage com o sistema.
- São bastante orientados ao domínio do problema.

Segurança e Requisitos

- Segurança não se resume a adicionar serviços/funcionalidades especializadas: SSL, HTTPS, firewall, etc.
- Segurança é um propriedade emergente do sistema em desenvolvimento.
- Segurança é resultado de muitos fatores.
- Especificação de requisitos não deve ser apenas uma lista de funcionalidades.

Caso de Uso

- Um típico caso de uso inclui:
  - Ator;
  - Pré-condições
  - Fluxo principal;
  - Fluxos alternativos;

Misuse & Abuse Case

- É um cenário dentro de um caso de uso no qual temos um ator se comportando de maneira inadequada.
- É um fluxo de eventos, como no caso de uso, mas com fins maliciosos.
- Importante:
  - Não levar código em conta aqui. O foco é o domínio.
  - Atores maliciosos são criativos.
  - Questionar o que o sistema assume como certo.
  - Mais foco no que *pode fazer* do que no que *fará.*

Misuse & Abuse Case

- Em resumo: pensar no que pode ser feito de errado no sistema!

Misuse vs Abuse

- Misuse é não intencional.
- Abuse é intencional.
- Misuse cases ainda são uma questão de segurança (oportunidade faz o ladrão).
- Abuse cases têm implicitamente a ideia de que o atacante está procurando vulnerabilidades.

Caso de uso: exemplo

- Cadastro de Interações medicamentosas:
  - Ator: Administrador do hospital.
  - Pré-condição: administrador foi autenticado.
  - Fluxo principal:
    - Admin escolhe dois medicamentos diferentes do cadastro.
    - Admin cadastra a dosagem mínima para cada um deles.
    - Admin insere texto breve sobre consequências potenciais da interação.
    - Sistema retorna uma tabela com os pacientes que podem ter esta interação.
    - Admin salva a interação cadastrada.

Misuse case: exemplo

- Passos 1 a 3 do fluxo principal são executados.
- A lista de pacientes apresentada ao admin inclui registros que ele não poderia acessar.
- Dano causado: privacidade do paciente foi violada.

Abuse case: exemplo

- Ator: atacante que conseguiu entrar no sistema como administrador.
- Fluxo:
  - Fluxo principal é repetido 1000 vezes.
    - Muitas interações são cadastradas para todos os medicamentos disponíveis.
    - Notas vagas e auto-geradas são inseridas para cada nova interação cadastrada.
  - Para quando o passo anterior leva mais de um minuto para ser executado.

Abuse case: exemplo

- Dano causado:
  - Pacientes são “soterrados” por alertas e começam a ignorá-los.
  - A disponibilidade do sistema é comprometida.

Exercício

- Leia a reportagem disponível no link: https://tecnoblog.net/noticias/2022/05/19/nubank-tinha-falha-de-seguranca-que-facilitava-roubo-de-dinheiro-usando-o-gmail/
- Escreva o caso de uso de recuperação de senhas conforme relatado na reportagem.
- Escreva o abuse case relacionado.

Exercício

- Leia a reportagem disponível no link: https://www.otempo.com.br/economia/golpe-da-maquininha-de-cartao-inventado-no-brasil-ganha-o-mundo-saiba-qual-1.3282530.
- Escreva o caso de uso de pagamento por cartão por aproximação.
- Escreva o abuse case relacionado.

Criando abuse cases

- Exige um pouco de experiência sobre ataques.
- O processo pode ser iniciado com um brainstorming.
- Começar analisando as interfaces de usuário e como elas podem ser usadas em ataques.
- Pensar em outros pontos que o atacante pode explorar:
  - Ler arquivos na máquina, gravar arquivos na máquina, grampear a comunicação, se comunicar diretamente com o banco, etc.

Misuse & abuse case

- Dificuldade:
  - casos tem potencial infinito de crescimento.
  - Necessário saber a hora de parar e limitar escopo.
- Concentrar-se em pontos típicos de ataque:
  - Valores próximos aos limites, comunicação entre sistemas e componentes, suposições usadas para construir o sistema.
  - Ações que analistas **assumem** que usuário não pode ou não vai fazer.
- Priorizar...

Requisitos de segurança

- Podem ser levantados a partir dos abuse/misuse cases.
- Utilizam os cenários descritos para definir requisitos para o sistema como um todo.

- Requirement 1: The system is required to have strong authentication measures in place at all system gateways and entrance points (maps to Goals 1 and 2).
- Requirement 2: The system is required to have sufficient process-centric and logical means to govern which system elements (e.g., data, functionality) users can view, modify, and/or interact with (maps to Goals 1 and 2).

Abuse cases - OWASP

- Conteúdo interessante:
  - https://cheatsheetseries.owasp.org/cheatsheets/Abuse_Case_Cheat_Sheet.html
  - Aqui, a OWASP oferece um guia prático para começar a projetar um processo para levantamento de abuse cases.
  - Ela pode ser aplicada também para métodos ágeis.

Análise de Riscos Arquitetural

Introdução

- Projeto e arquitetura do software:
  - Boas decisões de segurança aqui vão produzir um software mais resistente a ataques e apoiar o desenvolvimento do código de maneira mais segura,
- Análise de riscos é o caminho: catalogar ameaças, avaliar riscos, e planejar mitigação.

Introdução

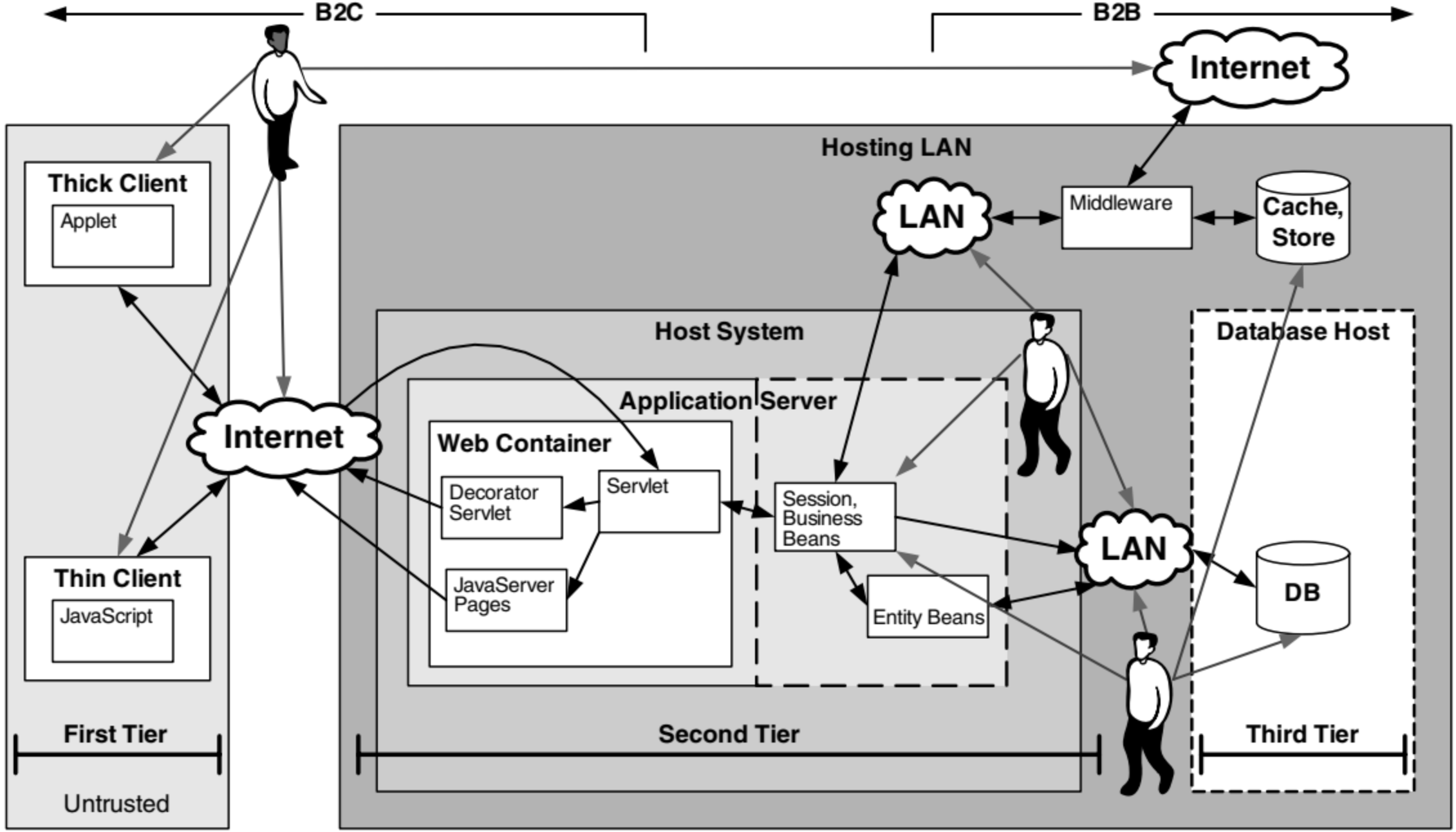
- Mais focado em discutir riscos logo depois de definir a arquitetura.
- Motivação: algumas decisões tomadas no início podem ajudar bastante:
  - Uso de criptografia.
  - Preocupações relacionadas a controle de acesso e autenticação.
  - Escolha das tecnologias utilizadas (lembrem-se: bibliotecas externas e plataformas de execução podem trazer vulnerabilidades).

Metodologia de análise de risco

- Seis atividades principais:
  - Caracterização do software.
  - Análise das ameaças.
  - Avaliação das vulnerabilidades da arquitetura.
  - Definição da probabilidade do risco.
  - Definição do impacto do risco.
  - Plano de mitigação de risco.

Caracterização do software

- Estratégias para executar esta fase podem variar.
- Um diagrama com uma visão geral do sistema tende a funcionar bem.
- Alguns artefatos que podem ajudar a fazer esta caracterização: requisitos funcionais e não funcionais, requisitos de arquitetura, documentos de casos de uso, abuse/misuse cases, diagramas com visões lógicas e físicas do sistema, etc.


B2C
B2B
Internet
Hosting LAN
Thick Client
Applet
LAN
Middleware
Cache, Store
Host System
Database Host
Application Server
Internet
Web Container
Decorator Servlet
Servlet
Session, Business Beans
LAN
DB
JavaServer Pages
Entity Beans
Thin Client
JavaScript
First Tier
Second Tier
Third Tier
Untrusted

Análise de ameaças

- Ameaças são agentes que violam a proteção da informação e as políticas de segurança.
- A análise de ameaças identifica estas ameaças para um dado cenário.
- Durante a análise, as ameaças são mapeadas a vulnerabilidades.
- Contramedidas para reverter estas vulnerabilidades fazem parte do plano.

Análise de ameaças

- Ameaças externas estruturadas: geradas por entidades organizadas ou patrocinadas por um estado.
- Ameaças transnacionais: geradas por organizações como células terroristas, cartéis de narcotráfico, etc.
- Ameaças externas não estruturadas: indivíduos conhecidos como *crackers*.

Avaliação de vulnerabilidades

- Resistência a ataques
  - Analisar o sistema sob a perspectiva de más e boas práticas conhecidas (exemplo no próximo slide).
  - Analisar o sistema sob a perspectiva de ataques bem conhecidos. Usar a CAPEC (*Common Attack Pattern Enumeration and Classification*), por exemplo, ou as listas da OWASP.

Avaliação de vulnerabilidades

- Principles for Software Security:
  - Least Privilege
  - Failing Securely
  - Securing the weakest link
  - Defense in depth
  - Separation of privilegie
  - Economy of mechanism
  - Least common mechanism
  - Reluctance to trust
  - Never assuming that your secrets are safe
  - Complete mediation
  - Psychological acceptability
  - Promoting privacy

Exercício

1- Escolha um dos *Principles for Software Security*.

2- Leia a descrição deste princípio e escreva um parágrafo, em português, explicando-o com as suas palavras.

3- Cada estudante vai apresentar o seu princípio para o restante da sala.

Avaliação de vulnerabilidades

- Análise de dependências
  - Incluir sistemas operacionais, rede, plataformas (WebLogic, PHP, Apache Tomcat).
  - Ideia é listar todas as vulnerabilidades que podem existir nestes componentes.
  - Catálogos como o CVE podem ser usados.
  - Abuse/misuse cases.

Avaliação de vulnerabilidades

- Mapear ameaças e vulnerabilidades:
  - Podemos usar modelos como o Microsoft STRIDE (falaremos dele mais tarde).
  - Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, e Elevation of Privilege.
  - Detalha as principais categorias onde temos eventos caracterizados por ameaças explorando vulnerabilidades.

Definição de probabilidade de risco

- Probabilidade: estimativa quantitativa de que o ataque tenha sucesso.
- Útil para priorizar riscos e avaliar as mitigações a serem aplicadas.
- O que é levado em conta:
  - Motivação da ameaça e sua capacidade.
  - Impacto da vulnerabilidade.
  - O grau de efetividade dos controles de segurança atuais

Definição de probabilidade de risco

High	The three qualities are all weak. A threat is highly motivated and sufficiently capable, a vulnerability exists that is severe, and controls to prevent the vulnerability from being exploited are ineffective.
Medium	One of the three qualities is compensating, but the others are not. The threat is perhaps not very motivated or not sufficiently capable, the controls in place might be reasonably strong, or the vulnerability might be not very severe.
Low	Two or more of the three qualities are compensating. The threat might lack motivation or capability; strong controls might be in place to prevent, or at least significantly impede, the vulnerability from being exploited; and the vulnerability might have a very low impact.

Definição do impacto de risco

- Quais são os impactos deste risco para o negócio?
- Se o ataque ocorre com sucesso, quais são as consequências para a organização?
- Importante:
  - Identificar recursos afetados.
  - Identificar o impacto no negócio.

Probabilidade vs impacto

		Impact		
		Low	***Medium***	***High***
Likelihood	***Low***	Low	Low	Medium
	Medium	Low	Medium	High
	High	Medium	High	High

UEL

Mitigação de risco

- Mitigação: mudar o cenário para diminuir a probabilidade do risco.
- Algumas ações possíveis:
  - Implementar controles de segurança mais rígidos (exemplo: mudar autenticação simples para autenticação de 2 fatores).
  - Mudar parâmetros do sistema (exemplo: tempo em que uma sessão de usuário permanece ativa).
- Sempre há um custo envolvido em ações de mitigação.

Exercício rápido

- Com base na fundamentação teórica que acabamos de ver, monte um processo para análise de risco de um software. Faça uma lista de passos e descreva rapidamente cada um deles.

Threat Modeling

- Ferramenta para análise de risco proposta pela Microsoft.
- Utiliza o modelo STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, e Elevation of Privilege).

STRIDE -> Propriedades de segurança

- Tampering -> violação de integridade
- Repudiation -> violação da integridade de informações passadas. Uma ameaça à crença na integridade.
- Information disclosure -> violação da confidencialidade.
- Denial of service -> violação da disponibilidade.
- Spoofing -> violação de autenticação.
- Elevation of privilege -> violação da autorização.

STRIDE

STRIDE Element	*Description*	*Example*
Spoofing	Tricking a system into believing a falsified entity is a true entity	Using stolen or borrowed credentials to log on as another nurse
Tampering	Intentional modification of a system in an unauthorized way	Changing patient data to incorrect values
Repudiation	Disputing the authenticity of an action taken	Denying that a prescribed treatment has been provided to the patient
Information Disclosure	Exposing information intended to have restricted access levels	Health data is sent over an unencrypted Bluetooth connection
Denial of Service (DoS)	Blocking legitimate access or functionality of a system by malicious process(es)	A Bluetooth SpO2 sensor is flooded with bad pairing requests, preventing legitimate connections
Elevation of Privilege (EoP)	Gaining access to functions to which an attacker should not normally have access according to the intended security policy of the product	A patient uses a web portal vulnerability to see all patient data, rather than their own

Playbook for threat modeling medical devices.
Link: https://www.mitre.org/publications/technical-papers/playbook-threat-modeling-medical-devices

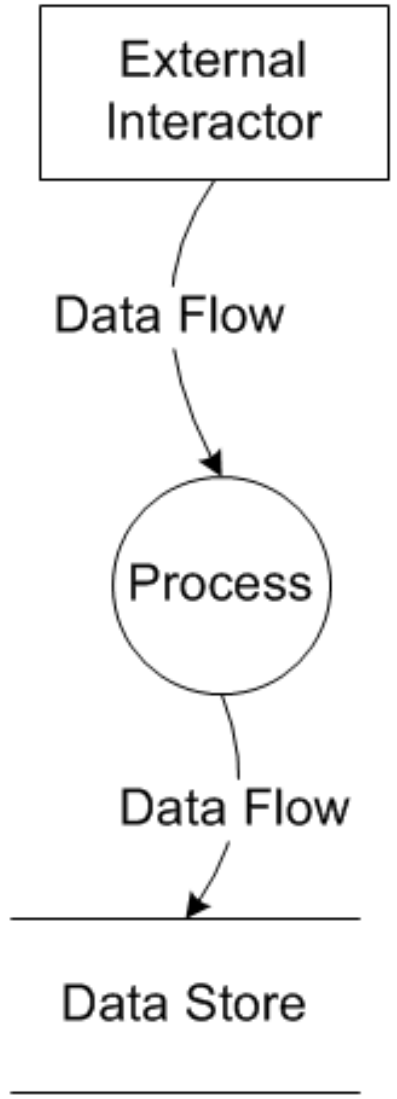
UEL

Threat Modeling

- Metodologia:
  - Define alguns elementos da arquitetura: processos, atores externos, *data stores*.
  - Estabelece os fluxos de dados utilizando os elementos.
  - Define fronteiras/limites de confiança dentro do sistema.
  - Mapeia STRIDE para cada elemento e relacionamento.

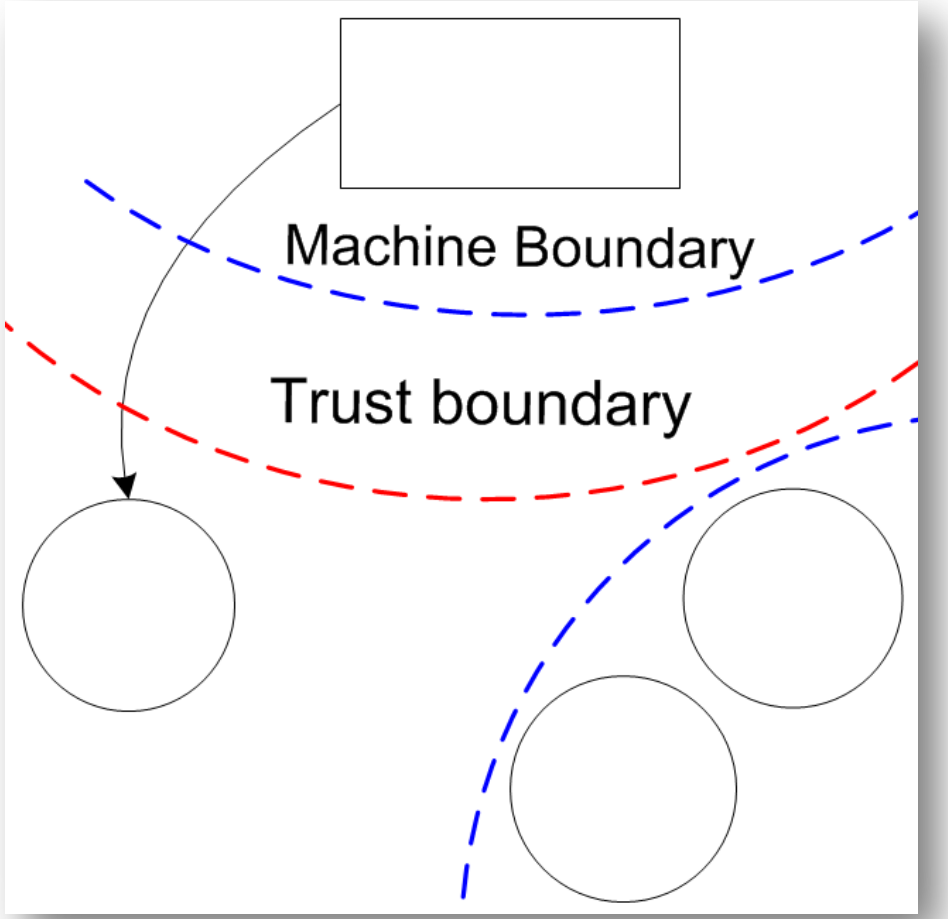
Primitivas

- Atores externos (*external interactor*)*:*
  - Clientes, outros sistemas dependências.
- Processos:
  - Funcionalidade importante para a arquitetura como um *dispatcher* ou um validador de entrada.
- Data store:
  - Banco de dados, sistema de arquivos.
- Fluxo de dados:
  - HTTP login request



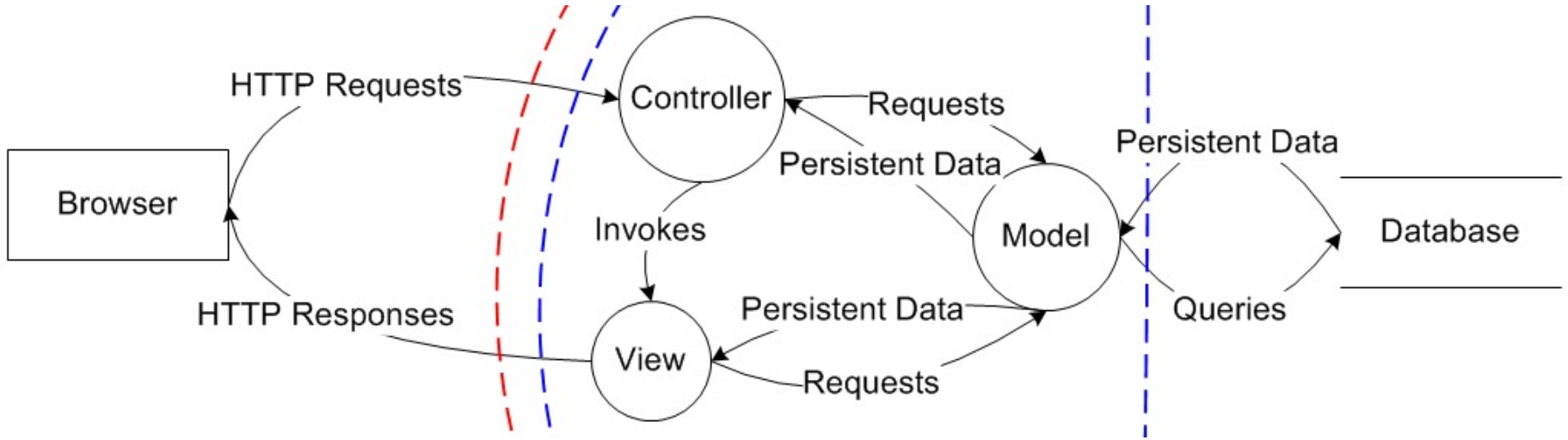
Boundaries

- Machine boundaries: dentro da mesma máquina.
- Trust boundaries: se a entrada é confiável ou não.



UEL

Exemplo: Aplicação MVC Genérica



- e.g. Como prevenir um spoofing por parte do cliente (navegador)?
- e.g. Como prevenir que as consultas sejam modificadas (tampering)?
- e.g. Como evitar vazamento de dados do banco de dados?

UEL

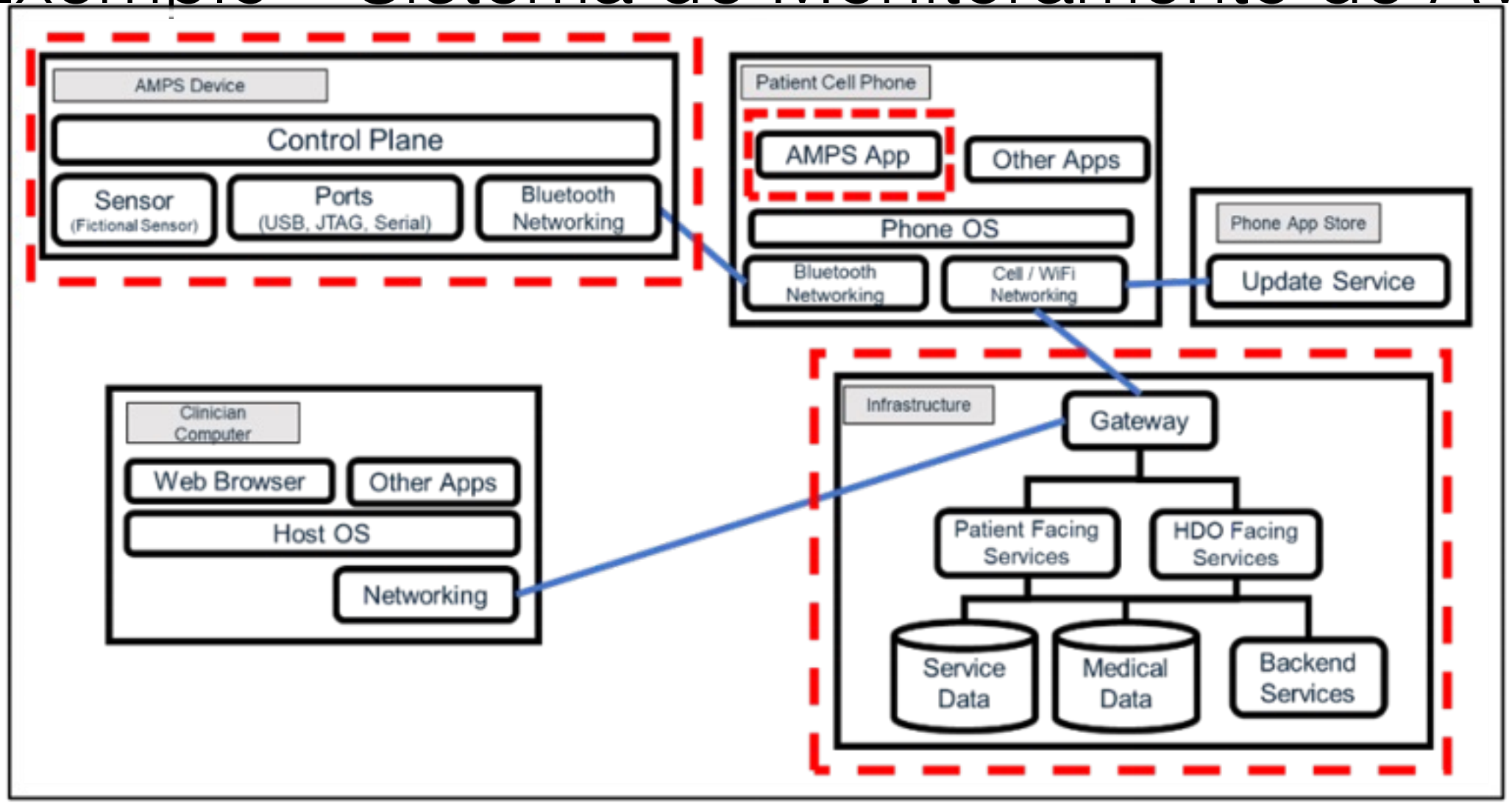
STRIDE -> DFD

Element	Spoof	Tamper	Repudiate	Info Disclosure	DoS	EoP
External Entity	X		X			
Process	X	X	X	X	X	X
Data Store		X	?	X	X	
Dataflow		X		X	X	

Playbook for threat modeling medical devices.
Link: https://www.mitre.org/publications/technical-papers/playbook-threat-modeling-medical-devices

UEL

Exemplo – Sistema de Monitoramento de AVC



Playbook for threat modeling medical devices.
Link: https://www.mitre.org/publications/technical-papers/playbook-threat-modeling-medical-devices

Exemplo – Sistema de Monitoramento de AVC

AMPS Component	Spoof	Tamper	Repudiate	Info	DoS	EoP
AMPS Device	1	2			3, 34, 35	4
AMPS App	5, 36	6		7	8	9, 37
App Store	10, 38	11		12	13	14
AMPSCS	15, 39, 40	16,41	17, 42, 43, 44	18, 45	19, 46, 47, 48	20, 49, 50
Clinician Computer	21, 51, 52	22		23	24, 53	25
Dataflow: Bluetooth				26	27, 54	
Dataflow: Cell/Wi-Fi Network		28		29	30	
Dataflow: Clinician Computer Internet		31		32	33	

Playbook for threat modeling medical devices.
Link: https://www.mitre.org/publications/technical-papers/playbook-threat-modeling-medical-devices

UEL

Exemplo – Sistema de Monitoramento de AVC

Reference ID	STRIDE Type	Description
1	Spoof	An attacker could pretend to be an authorized phone app to obtain readings from the device
2	Tamper	Control plane could be attacked and given incorrect readings
3	DoS	Invalid input could cause device to crash
4	EoP	Device could be hacked, and software could be installed to perform other actions (such as make it part of a botnet, enable lateral movement, etc.)
34	DoS	Software could be corrupted
35	DoS	Battery could be drained more rapidly than normal

Playbook for threat modeling medical devices.
Link: https://www.mitre.org/publications/technical-papers/playbook-threat-modeling-medical-devices

Threat Modeling: Resumo

- What are we working on?
  - Construir diagrama que ofereça uma visão geral do sistema.
- What can go wrong?
  - Identificar ameaças com o modelo STRIDE.
- What are we going to do about it?
  - Trabalhar em plano de mitigação, reduzindo os riscos.

Codificação Segura

Introdução

- Como podemos ter mais segurança na hora de escrever código?
- Há ferramentas que ajudam nesse processo?
- Há catálogos ou guias com boas práticas para as diferentes linguagens?

Análise de Código

- Lembrando:
  - Flaws são problemas que se apresentam no código, mas normalmente são consequência de um design problemático.
  - Bugs são problemas introduzidos durante a implementação do código.
- Nesta parte do curso, vamos nos focar em bugs de implementação.

Vulnerabilidades de segurança comuns

- Validação de entrada incompleta ou incorreta:
  - Dá margem a ataques baseados em buffer overflow, cross-site scripting, integer overflow, envenenamento de cache, SQL injection, e OS command injection.
  - Riscos à integridade e confidencialidade.
  - Técnicas para defesa: implementação de *white lists* ou *black lists*, de forma a tratar estas entradas.
  - https://owasp.org/Top10/A03_2021-Injection/

Vulnerabilidades de segurança comuns

- SQL Injection:
  - Consultas SQL utilizam, frequentemente, parâmetros passados por usuários.
  - Atacantes podem manipular estes parâmetros para acessar a base de dados.
  - Em casos mais graves, atacantes podem até mesmo executar código em nível de SO (stored procedures).

Vulnerabilidades de segurança comuns

- Vídeos sobre Injection e XSS:
  - XSS: https://www.youtube.com/watch?v=S_MPsbWASt0
  - SQL Injection: https://www.youtube.com/watch?v=uijotyGm8W4

Vulnerabilidades de segurança comuns

- Buffer overflows:
  - Permite que seja injetado código malicioso em um alvo.
  - Linguagens que não checam automaticamente ponteiros e limites de estruturas de dados (por exemplo, C) estão mais expostas.
  - Consequências são imprevisíveis.
  - https://cwe.mitre.org/data/definitions/121.html

Vulnerabilidades de segurança comuns

- Exceções:
  - Situações em que o código se desvia de seu curso “normal”.
  - Divisão por zero, violação de proteção de memória, etc.
  - Algumas linguagens disponibilizam *exception handlers*.
  - Estas situações precisam sempre ser verificadas e tratadas.
  - Procure capturar as exceções para tratá-las em um determinado ponto do sistema.

Vulnerabilidades de segurança comuns

- Race conditions:
  - Múltiplas threads sincronizadas de maneira inadequada.
  - Alguns exemplos: loops infinitos, deadlocks, falha em sincronizar acesso a recursos compartilhados.
  - Há diversas soluções para evitar estes problemas: mutex, semáforos, etc.
  - Procure evitar suposições pobres: “não há possibilidade de haver mais de uma thread aqui”...
  - Sempre que houver concorrência, desconfie de possíveis consequências indesejadas.

Vulnerabilidades de segurança comuns

- Outros exemplos de problemas comuns:
  - Exposição de dados sensíveis: https://www.youtube.com/watch?v=83GV2ntqK-E
  - Entidades Externas de XML: https://www.youtube.com/watch?v=GDpEebVLvD8

Revisão de código fonte

- Uma das práticas que já é frequentemente utilizada.
- Prática que pode ser facilmente integrada ao desenvolvimento de código.
- Uso de convenções de codificação e check-lists.
- Normalmente:
  - temos o papel de um revisor de código.
  - uso de ferramentas de análise para automatizar parte da tarefa.

Ferramentas de Análise Estática de Código

- A análise **estática** é realizada antes que o código seja compilado.
- Ferramentas de análise estática fazem uma varredura no código procurando por erros que não são detectados pelo compilador.
- É comum que estas ferramentas apresentem relatórios gráficos sobre os erros e ajude a geri-los.

Ferramentas de Análise Estática de Código

- Estas ferramentas não devem ser encaradas como a solução de todos os problemas.
- Quando uma ferramenta não encontra nenhum erro, isso não quer dizer que o software é 100% seguro.
- A análise está sujeita a falsos positivos e falsos negativos.
- A grande vantagem é a capacidade de identificar erros comuns.

Ferramentas de Análise Estática de Código

- Problemas comuns detectados em análise estática:
  - Trecho de código inatingível.
  - Variáveis não declaradas.
  - Variáveis não inicializadas.
  - Argumentos incompatíveis com os tipos dos parâmetros.
  - Funções e procedimentos que não são nunca invocados.
  - Funções cujos resultados não são utilizados.
  - Violação de limites de vetores.
  - Uso inadequado de ponteiros.

Ferramentas de Análise Estática de Código

- Problemas que podem não ser detectados por análise estática:
  - Bibliotecas de criptografia mal projetadas.
  - Algoritmos inadequados para a situação.
  - Senhas embutidas no código fonte.
  - Etc.

Métricas

- Derivar métricas que podem apoiar a análise do código.
- Dados quantitativos sobre o código são coletados:
  - Número de linhas de código, número de brechas de segurança, número de inconformidades por severidade, etc.
- A partir de dados coletados, podemos derivar métricas:
  - Número de brechas por linhas de código.
- Pode-se determinar metas para estas métricas.
- É possível acompanhar a evolução das métricas e avaliar melhorias.

Práticas de codificação

- Práticas de codificação descrevem métodos, técnicas, processos, ferramentas e bibliotecas que podem prevenir a exploração de vulnerabilidades.
- Normalmente, são direcionadas a cada linguagem.
- Grande parte das vulnerabilidades documentadas são derivadas de erros comuns de programação.

Práticas de codificação

- https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards
- https://www.oracle.com/java/technologies/javase/seccodeguide.html

Práticas de codificação

- Alguns princípios genéricos de codificação defensiva:
  - Evite código duplicado, pois ele é mais difícil de proteger.
  - Conheça bem as APIs e códigos que está usando (evite copiar código da Internet sem entender).
  - Fique atento a novas vulnerabilidades (CVE) que são catalogadas e às fragilidades expostas no CWE.

Referências bibliográficas

- G. McGraw, “Software Security: Building Security in”. Addison Wesley Professional, 2006.
- J. Allen et al., “Software Security Engineering: a Guide for Project Managers”. Addison Wesley Professional, 2008.
- E. T. Nakamura e P. L. de Geus, “Segurança de Redes em Ambientes Cooperativos”, Ed. Novatec, 2007.
- W. Stallings, “Criptografia e Segurança de Redes”, Ed. Pearson, 4. ed., 2008.

UEL